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 shortcoming 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 \( \mathbf{P}_i = (p_{i1}, p_{i2}, ..., p_{iN}) \). For the global variant, the best previously visited position of the swarm is \( \mathbf{G} = (g_{1}, g_{2}, ..., g_{N}) \), and let the superscripts denote the iteration number, then each particle is manipulated according to the following two equations [17]:

\[
\begin{align*}
v_{ij}^{t+1} &= wv_{ij}^t + c_1 r_1^t (p_{ij}^t - x_{ij}^t) + c_2 r_2^t (g_{ij}^t - x_{ij}^t) \\
x_{ij}^{t+1} &= x_{ij}^t + v_{ij}^{t+1}
\end{align*}
\]

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).

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

$$k = 2\sqrt{2 - \phi - \sqrt{\phi^2 - 4\phi}}, \text{where } \phi = c_1 + c_2, \phi > 4$$

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):

$$v_{ij}^{t+1} = v_{ij}^t + M_{ij}^t$$

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

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

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

$$A_i^t = \frac{fit^t_i - Maxfit^t}{Medfit^t - Maxfit^t}$$

$$a_i^t = A_i^t \sum_{j=1}^{N} A_j^t$$

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. \( gb = \text{findBest}(x); //to find the best agent in population\)
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>
<thead>
<tr>
<th>fun</th>
<th>CPSO</th>
<th>MPSO</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Best</td>
<td>Median</td>
</tr>
<tr>
<td>F3</td>
<td>7.55E-15</td>
<td>1.78</td>
</tr>
<tr>
<td>F4</td>
<td>0</td>
<td>3.45E-2</td>
</tr>
<tr>
<td>F9</td>
<td>1.57E-32</td>
<td><strong>2.03E-31</strong></td>
</tr>
<tr>
<td>F10</td>
<td>1.35E-32</td>
<td><strong>1.22E-31</strong></td>
</tr>
</tbody>
</table>
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( )

1. PD = 1
2. While ( rand(0,1) > p ) and ( PD < D )
   1. PD = PD + 1
3. End While
4. Return PD

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

<table>
<thead>
<tr>
<th>PD</th>
<th>F1</th>
<th>F7</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Best</td>
<td>Worst</td>
</tr>
<tr>
<td>1</td>
<td>4.77E-118</td>
<td>8.99E-109</td>
</tr>
<tr>
<td>5</td>
<td>5.64E-199</td>
<td>8.19E-196</td>
</tr>
<tr>
<td>10</td>
<td>1.68E-273</td>
<td>1.29E-269</td>
</tr>
<tr>
<td>15</td>
<td>1.0E-323</td>
<td>7.07E-318</td>
</tr>
<tr>
<td>20</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>25</td>
<td>0</td>
<td>4.0E-322</td>
</tr>
<tr>
<td>30</td>
<td>2.79E-243</td>
<td>1.36E-207</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>P</th>
<th>F1</th>
<th>F4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Best</td>
<td>Worst</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>4.4E-323</td>
</tr>
<tr>
<td>20</td>
<td>1.2E-175</td>
<td>2.1E-165</td>
</tr>
<tr>
<td>30</td>
<td>1.5E-121</td>
<td>6.4E-108</td>
</tr>
<tr>
<td>40</td>
<td>8.12E-87</td>
<td>4.81E-80</td>
</tr>
<tr>
<td>50</td>
<td>1.25E-70</td>
<td>8.60E-64</td>
</tr>
<tr>
<td>60</td>
<td>1.37E-56</td>
<td>1.03E-51</td>
</tr>
<tr>
<td>70</td>
<td>4.04E-48</td>
<td>4.68E-44</td>
</tr>
<tr>
<td>80</td>
<td>2.02E-41</td>
<td>1.56E-37</td>
</tr>
<tr>
<td>90</td>
<td>3.32E-37</td>
<td>6.87E-33</td>
</tr>
<tr>
<td>100</td>
<td>1.51E-33</td>
<td>2.07E-29</td>
</tr>
</tbody>
</table>
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. \( gb = \text{findBest}(x); \) //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. \( PD = \text{producePD}( ) \)
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 \( (e < 0 \text{ OR random()} < \exp(-e/t) ) \) 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 \( (x[i] \text{ is better than } pb[i]) \) Then \( pb[i] = x[i]; \)
34. If \( (x[i] \text{ is better than } gb) \) 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

\[
f_1(x) = \sum_{i=1}^{n} x_i^2, \quad |x_i| \leq 100
\]
(2) Generalized Rosenbrock’s Function

\[ f_2(x) = \sum_{i=1}^{n-1} (100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2), \quad |x_i| \leq 30 \]

(3) Ackley’s Function

\[ f_3(x) = -20 \exp(-0.2 \sqrt{\frac{1}{n} \sum_{i=1}^{n} x_i^2}) - \exp\left(\frac{1}{n} \sum_{i=1}^{n} \cos(2\pi x_i)\right) + 20 + e, \quad |x_i| \leq 32 \]

(4) Generalized Griewank’s Function

\[ f_4(x) = \frac{1}{4000} \sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1, \quad |x_i| \leq 600 \]

(5) Generalized Rastrigrin’s Function

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

(6) Generalized Schwefel’s Problem 2.26

\[ f_6(x) = \sum_{i=1}^{n} -x_i \sin(\sqrt{|x_i|}), \quad |x_i| \leq 500 \]

(7) Salomon

\[ f_7(x) = -\cos\left(2\pi \sqrt{\sum_{i=1}^{n} x_i^2}\right) + 0.1 \sqrt{\sum_{i=1}^{n} x_i^2} + 1, \quad |x_i| \leq 100 \]

(8) Whitely

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

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

(9) Generalized Penalized Function 1

\[ f_9(x) = \frac{\pi}{n} \left[ 10 \sin^2(\pi y_{i+1}) + \sum_{i=1}^{n} (y_{i+1} - 1)^2 (1 + 10 \sin^2(\pi y_{i+1})) + (y_{i+1} - 1)^2 \right] \]

\[ + \sum_{i=1}^{n} u(x_i, 10, 100, 4), \quad |x_i| \leq 50 \]

\[ y_{i+1} = 1 + \frac{1}{4} (x_i + 1) \]

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

(10) Generalized Penalized Function 2
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 $PD$.

### 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.

---

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

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

**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
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