Applied Soft Computing xxx (2012) xxx–xxx

Contents lists available at SciVerse ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

A hierarchical particle swarm optimizer with latin sampling based memetic algorithm for numerical optimization

Yong Peng[a,*], Bao-Liang Lu[a,b]

[a] *Center for Brain-like Computing and Machine Intelligence, Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai 200240, PR China*
[b] *MoE-Microsoft Key Laboratory for Intelligent Computing and Intelligent Systems, Shanghai Jiao Tong University, Shanghai 200240, PR China*

A R T I C L E I N F O

Article history:
Received 31 December 2011
Received in revised form 13 May 2012
Accepted 16 May 2012
Available online xxx

Keywords:
Memetic algorithm
Particle swarm optimizer
Latin hypercube sampling
Comprehensive learning
Cylindricity

A B S T R A C T

Memetic algorithms, one type of algorithms inspired by nature, have been successfully applied to solve numerous optimization problems in diverse fields. In this paper, we propose a new memetic computing model, using a hierarchical particle swarm optimizer (HPSO) and latin hypercube sampling (LHS) method. In the bottom layer of hierarchical PSO, several swarms evolve in parallel to avoid being trapped in local optima. The learning strategy for each swarm is the well-known comprehensive learning method with a newly designed mutation operator. After the evolution process accomplished in bottom layer, one particle for each swarm is selected as candidate to construct the swarm in the top layer, which evolves by the same strategy employed in the bottom layer. The local search strategy based on LHS is imposed on particles in the top layer every specified number of generations. The new memetic computing model is extensively evaluated on a suite of 16 numerical optimization functions as well as the cylindricity error evaluation problem. Experimental results show that the proposed algorithm compares favorably with conventional PSO and several variants.

1. Introduction

Optimization has been a research hotspot for several decades. Many real-world optimization problems in engineering are becoming increasingly complicated, so optimization algorithms with high performance are needed [1,2]. Unconstrained optimization problems can be formulated as D-dimensional optimization problems over continuous space

$$min\, f(\mathbf{x}),\ \mathbf{x} = [x_1, x_2, \ldots, x_D] \tag{1}$$

Evolutionary algorithms, inspired by natural evolution, have been widely used as effective tools to solve optimization problems. One class of nature inspired algorithms are swarm intelligent algorithms. Particle swarm optimizer (PSO) [3,4] has attracted attention in the academic and industrial community. Although PSO shares many similarities with evolutionary algorithms, the original PSO does not use the traditional evolution operators such as crossover and mutation. PSO draws on the swarm behavior of birds flocking where they search for food in a collaborative way. Each member, in the swarm, called a *particle*, represents a potential solution to the target problem and it adapts its search patterns by learning from its own experience and other members' experience. The particle is a point in the search space and it aims at finding the global optimum which is regarded as the location of food. Each particle has two attributes called *position* and *velocity* and its direction of flight is adjusted according to the experiences of the swarm. The swarm as a whole searches for the global optimum in D-dimensional feasibility space.

The PSO algorithm is easy to understand and implement, and has been proved to perform well on many optimization problems. However, it may easily get trapped in a local optimum for many reasons, such as the lack of diversity among particles and overlearning from the best particle found so far. To improve PSO's performance on complex numerical optimization problems, we propose a hierarchical PSO framework, in which several swarms evolve in parallel towards the global optimum and we design a new mutation operator to increase the diversity of swarms. After evolving for a specified number of generations, a latin hypercube sampling method is used to execute the local search.

This paper is organized as follows. Section 2 introduces the original PSO and some variants. Section 3 describes the proposed hierarchical PSO with latin sampling based memetic algorithm, including four subsections: hierarchical PSO framework, mutation strategy, latin hypercube sampling based local search strategy and the overall framework of the proposed memetic algorithm. Section 4 gives the experimental results, describes the related parameter tuning process and compares the performance of the proposed algorithm on a suite of test problems to that of other PSO variants. Section 5 gives conclusions and describes future work.

* Corresponding author.
E-mail address: StanY.Peng@gmail.com (Y. Peng).

1568-4946/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.asoc.2012.05.020

```
//Initialize swarm S
for $i := 1$ to swarmsize do
    for $d := 1$ to $D$ do
        $V_i^d := rand[V_{min}, V_{max}]$; $X_i^d := rand[X_{min}, X_{max}]$;
    end for
end for
Compute the fitness value of each particle $F = (f_1, f_2, \ldots, f_{ps})$;
Set the pbest $= (pbest_1, pbest_2, \ldots, pbest_{ps})$ and the $gbest$;
Set the acceleration constants $c_1$ and $c_2$;
Set the iteration counter $t$:=0;
while $t \leq Gen$ do
    for $i := 1$ to swarmsize do
        for $d := 1$ to $D$ do
            //Update the velocity $V_i^d$ of particle $X_i$ using Eq.2
            $V_i^d := V_i^d + c_1 \cdot rand1_i^d \cdot (pbest_i^d - X_i^d) + c_2 \cdot rand2_i^d \cdot (gbest^d - X_i^d)$;
            //Update the position $X_i^d$ of particle $X_i$ using Eq.3
            $X_i^d := X_i^d + V_i^d$;
        end for
        Evaluate the fitness value $f_i$ of the new particle $X_i$;
        if $f_i$ is better than the fitness value of $pbest_i$ then
            Set $X_i$ to be $pbest_i$;
        end if
        if $f_i$ is better than the fitness value of $gbest$ then
            Set $X_i$ to be $gbest$;
        end if
    end for
    if termination condition is met then
        break;
    else
        $t := t + 1$;
    end if
end while
```

Fig. 1. Original particle swarm optimizer.

2. Particle swarm optimizers

2.1. Original PSO

PSO is a stochastic optimization algorithm which simulates swarm behavior. The individuals move over a specified *D*-dimensional feasible space. As in a the genetic algorithm, the particles in PSO are initialized with random velocities and positions. The algorithm adaptively updates the velocity and position of each particle in the swarm by learning from the good experiences. In the original PSO [3], the velocity $V_i{}^d$ and position $X_i{}^d$ of the *d*th dimension of the *i*th particle are updated as follows.

$$V_i^d := V_i^d + c_1 \cdot rand1_i^d \cdot (pbest_i^d - X_i^d) + c_2 \cdot rand2_i^d \cdot (gbest^d - X_i^d)$$
$$X_i^d := X_i^d + V_i^d \qquad (2)$$

where $X_i = (X_i^1, X_i^2, \ldots, X_i^D)$ is the position of the *i*th particle and $V_i = (V_i^1, V_i^2, \ldots, V_i^D)$ represents velocity of particle *i*, $\mathbf{pbest}_i = (pbest_i^1, pbest_i^2, \ldots, pbest_i^D)$ is the best previous position yielding the best fitness value for the *i*th particle, $\boldsymbol{gbest} = (gbest^1, gbest^2, \ldots, gbest^D)$ is the best position found so far over the whole swarm, c_1 and c_2 are the acceleration constants, reflecting the weighting of stochastic acceleration terms that pull each particle towards *pbest* and *gbest* positions, respectively. $rand1_i^d$ and $rand2_i^d$ are two random numbers in the range [0,1].

A particle's velocity on each dimension is confined to a maximum magnitude V_{max}. If $|V_i^d|$ exceeds a pre-specified positive constant value V_{max}^d, then the velocity on the dimension is assigned to $sign(|V_i^d|)V_{max}^d$.

The framework of the original PSO is shown in Fig. 1. From the flow of the iterative process, we can find that each particle flies to the global best particle in the swarm; this leads to a severe drawback of overlearning from the best particle. Consequently, the diversity of the whole swarm will drop down dramatically. If the best particle does not share the same niche with the global optimum, the particles may easily get trapped in a local optimum. Since PSO's introduction in 1995, many researchers have worked on improving its performance in various ways and many more effective variants have been proposed; these will be discussed in next subsection.

2.2. Some variants of PSO

This section gives a brief survey of several PSO variants proposed in recent years. Shi and Eberhart [5] introduced inertia weight *w* into the original PSO algorithm, so the criterion for updating the velocity was changed to

$$V_i^d := w \cdot V_i^d + c_1 \cdot rand1_i^d \cdot (pbest_i^d - X_i^d) + c_2 \cdot rand2_i^d \cdot (gbest^d - X_i^d). \qquad (3)$$

They indicated that the inertia weight plays an important role in balancing the global and local search abilities; a large inertia weight encourages global search while a small inertia weight encourages local search. Based on this idea, the inertia weight is usually set to decrease linearly over iterations.

Different types of topologies have been designed to improve PSO's performance in solving different optimization problems. Kennedy [6,7] claimed that PSO with a small neighborhood might perform better on complex problems, while PSO with large neighborhood would perform better on simple problems. Suganthan [8] defined the neighborhood of a particle as the several nearest particles in each iteration so that a dynamic neighborhood is computationally intensive. Jian et al. [9] examined several neighborhood topologies. The unified PSO (UPSO) proposed by Parsopoulos and Vrahatis [10] combined the global version and local version of the original PSO. Mendes et al. [11] used all the neighbors of the particle to update the velocity instead of the *pbest* and the *gbest*. The neighbors of each particle were selected based on its fitness value and the size of neighborhood. Peram et al. [12] proposed the fitness-distance-ratio-based PSO (FDR-PSO). When updating each velocity dimension, the FDR-PSO algorithm selects one other particle *nbest*, which has a higher fitness value and is nearer to the particle being updated. In comprehensive learning PSO (CLPSO) [13], the velocity of each dimension is influenced by *pbest* of every other particle, which increases the diversity of the swarm for multimodal optimization problems. In [14], several subswarms were used to coevolve with each other. The entire population was shuffled at periodic stages and subswarms were reassigned. Yang and Li [15] developed a hierarchical clustering method to partition the original swarm into several subswarms, which locate and track multiple optima in dynamic environments. Wang et al. [16] proposed a memetic algorithm based on a particle swarm optimizer with a ring-shaped topology; later, he improved his algorithm by partitioning particles in the ring-shaped topology structure into several species which can update information in parallel [17]. Chen [18] proposed a two-layer PSO (TLPSO) for unconstrained optimization problems, where each subswarm was made to evolve based on the original PSO.

Although the original PSO does not use the traditional evolution operators such as crossover and mutation, researchers introduced some other search techniques including evolutionary operators into PSO to improve its performance. Evolutionary operators such as crossover, mutation and selection were used in [19–21]. In Ref. [22], deflection, stretching and repulsion techniques are used to find as many minima as possible by preventing particles from moving to a previously discovered minimal region. Cooperative PSO (CPSO-H) [23] uses one-dimensional swarms to search each dimension separately. In recent years, many advanced operators have been introduced to improve PSO's performance. Ling et al. [24] employed a wavelet-theory-based mutation operation to enhance PSO in exploring the solution space more effectively. Zhao [25] proposed a perturbed PSO (pPSO) algorithm which introduced the perturbed global best to deal with the problem of premature convergence and diversity maintenance within the swarm. Gao et al. [26] incorporated the Henon map based mutation operator, which divided the mutation operator into global and local mutation operators; this enabled the particles to have a stronger exploration ability and fast convergence rate.

Although many variants of PSO have been proposed, all of which enhance the performance of original PSO to some extent, the effectiveness of these variants in dealing with diverse problems with different characteristics is still unsatisfying. For example, CLPSO's performance on ill-conditioned problems is poor and an algorithm [27] with high convergence speed is prone to shrink towards local optima. So taking measures including model structure, velocity updating strategy, and the hybrid operators simultaneously according to the particles' behavior to improve its performance may be a feasible path to get a satisfactory result over diverse numerical optimization problems.

3. The proposed memetic algorithm

In this section, we introduce the proposed memetic algorithm in detail; it is based on a hierarchical PSO framework and some search techniques including a local search strategy called the latin hypercube sampling method and a hybrid mutation strategy.

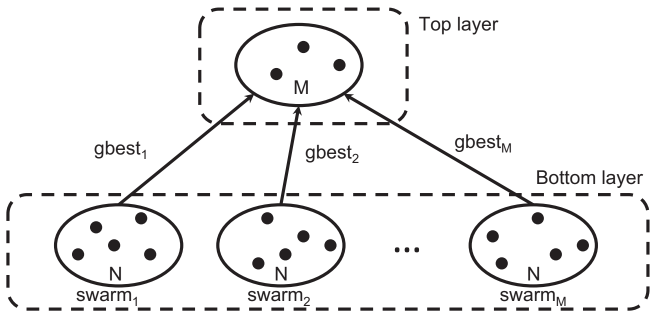

Fig. 2. The architecture of two-layer hierarchical PSO.

3.1. The hierarchical particle swarm optimizer

There are two versions of PSO, global and local, according to the approach of choosing *gbest*. In the global version, each particle can be influenced by the particle with best fitness in the whole swarm, which causes all the particles to move and converge quickly on one optimum point in the search space. By contrast, the local version only allows a particle to be influenced by the best fitness particle from its neighborhood, which makes the algorithm exhibit a good exploration capacity because the population can slowly converge to the optimal space. Recently, many algorithms have been proposed to partition the population into several subswarms based on Euclidean distance [28], fitness value [29] and some other metrics [15,17]. These subswarms are different definitions of the neighborhood, and each particle can only interact with particles in its neighborhood to avoid converging too fast. Obviously, computing the Euclidean distance is time-consuming when the dimension is high; individuals with similar fitness values which are prone to be classified into the same group may be in different niches. And the species formation method [17] is complicated and partially depends on the distance of particles. Here, we propose a two-layer hierarchical PSO model. There are *M* swarms in the bottom layer with *N* particles in each swarm and only one swarm in the top layer. Fig. 2 gives the architecture of the hierarchical PSO. For each swarm in the bottom layer, particles move towards the optimum based on the comprehensive learning method [13] described below, which is a typical local version of PSO. After each iteration, *M* swarms in the bottom layer will generate *M* best particles which will stand chances into the top layer. So in the top layer, the number of particles is identical to the number of swarms in the bottom layer and they are trained by comprehensive learning as well.

The reasons for the selecting hierarchical PSO can be stated as follows. First, several swarms evolving in parallel can have a good chance to reach the global optimum even if some of them stagnate in local optima. Second, the swarms are generated randomly which saves time in computing the neighborhood based on Euclidean distance. Though simple, this approach might be effective. For this model, the movement of particles in the bottom layer is similar to the local search and the movement of particles in the top layer is similar to the global search. The best particle in the top layer can influence particles in the bottom layer indirectly so that the speed of convergence will slow down. So this model can work for both exploitation and exploration simultaneously.

The comprehensive learning method [13] used to train particles in the hierarchical PSO modal, is specifically designed for complex multimodal problems. Simply speaking, CLPSO designs a set of exemplars $pbest^{d}_{fi(d)}$ for each particle to update its velocity instead of the traditional *pbest* and *gbest*, which enlarges the search scope and enhances the performance of local search. Fig. 3 gives the flow of the comprehensive learning method.

```
Initialize the swarm;
$V_{max} := 0.25 \cdot (X_{max} - X_{min})$; $w_0 := 0.9, w_1 := 0.2$;
for $k := 1$ to $Gen$ do
  $w(k) := \frac{(w_0 - w_1) \cdot (Gen - k)}{Gen}$;
  if Mod(k,10)==1 then
    for $i := 1$ to $swarmsize$ do
      $rc := randperm(D)$; $a_i := zeros(1, D)$; $b_i := zeros(1, D)$;
      $a_i(rc(1:m)) := 1$; $b_i = \lceil rand(1, D) - 1 + Pc \rceil$;
      $f_i := \lceil rand(1, D) \cdot swarmsize \rceil$;
    end for
  end if
  for $i := 1$ to $swarmsize$ do
    for $d := 1$ to $D$ do
      if $a_i^d == 1$ then
        $V_i^d := w_k \cdot V_i^d + rand() \cdot (gbest_d - V_i^d)$;
      else if $b_i^d == 1$ then
        $V_i^d := w_k \cdot V_i^d + rand() \cdot (pbest_{fi(d)}^d - V_i^d)$;
      else
        $V_i^d := w_k \cdot V_i^d + rand() \cdot (pbest_i^d - V_i^d)$;
      end if
      $V_i^d := min(V_{max}^d, max(-V_{max}^d, V_i^d))$;
      $X_i^d := X_i^d + V_i^d$;
    end for
    if for each $d$, $X_i^d \in [X_{min}, X_{max}]$ then
      Calculate the fitness value of $X_i$;
      Update $pbest$ and $gbest$;
    end if
  end for
  Stop if the termination condition is met;
end for
```

Fig. 3. Comprehensive learning PSO.

3.2. Mutation strategy

Most variants of PSO adopt strategies to update the old velocity vector based on the particles in neighborhood, so they have difficulty in adapting quickly to the different optimization stages of ill-conditioned problems. In this subsection, we propose a new mutation operator, inspired by the mutation operation in Differential Evolution (DE) [30]. It updates the particles' positions based on the differential information and the *pbest*. The mutation operator can be formulated as

$$X_i^d := c \cdot (X_k^d - X_j^d) + c \cdot (pbest_i^d - X_i^d); \quad c \sim N(0.5, 0.2); \tag{4}$$

where X_k^d and X_j^d are the dth variables of two randomly selected other particles, $N(0.5,0.2)$ represents the Gaussian distribution with mean 0.5 and standard deviation 0.2.

We carry out the mutation operation after updating the *pbest* and *gbest* in both the bottom layer and the top layer based on the probability *Pm* except the best particle in each swarm. This operator will generate a disturbance when particles' position are close to local optima.

3.3. Local search based on latin sampling

Latin hypercube sampling, which was proposed by Mckay [31], is a stratified sampling approach. This paper employs this sampling method to exploit the excellent subspace which has been found at present. Suppose that V is a hypercube with dimension n, of which each dimension x^i is denoted as $[x_l^i, x_u^i](i = 1, 2, \ldots, n, x_l^i$ and x_u^i are the lower bound and the upper bound of dimension i, respectively), then the algorithm of generating H samples in this hypercube V is Fig. 4.

Here, a simple instance is provided to demonstrate the Latin Sampling Process in detail. If the dimension of the hypercube is two and the sampling scale is eight, then a satisfactory sampling matrix A is formed.

$$A = \begin{bmatrix} 8 & 3 & 6 & 7 & 5 & 1 & 2 & 4 \\ 6 & 1 & 5 & 3 & 7 & 2 & 8 & 4 \end{bmatrix}^T \tag{5}$$

step1: Determine the sampling scale H;
step2: Partition the interval $[x_l^i, x_u^i]$ of each dimension into $H+1$ equivalent subintervals, that is

$$x_l^i = x_0^i < x_1^i < x_2^i < \ldots < x_j^i < x_{j+1}^i < \ldots < x_H^i = x_u^i$$

and as a result, the original hypercube is partitioned into H^n small sub hypercubes.
step3: Generate a $H * n$ matrix (termed the sampling matrix, denoted as A), each column of which is a random arrangement of array $[1, 2, \ldots, H]$;
step4: Each row of the sampling matrix A is a selected hypercube, and the H samples will be produced randomly from each selected hypercube.

Fig. 4. Latin sampling process in hypercube.

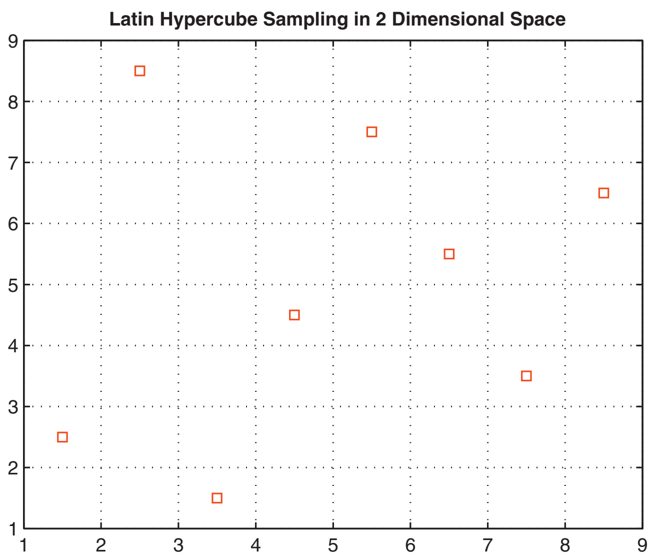

Fig. 5. Latin hypercube sampling in 2 dimensional space.

and the corresponding samples in the hypercube is showed in Fig. 5. Latin hypercube sampling can be viewed as a space-filling design, which means that one and only one sample is selected in each row or column of each sub hypercube. So the samples generated by latin sampling are distributed uniformly in the hypercube space and this is helpful to maintain the diversity of population.

3.4. *The proposed memetic algorithm*

In this section, we introduce the hierarchical PSO with latin hypercube sampling based memetic algorithm (MA-HPSOL) as whole. Fig. 6 gives the overall framework of the proposed algorithm.

From Fig. 6, we know that hierarchical PSO is the main framework of the proposed memetic algorithm. Swarms in the framework are trained by the comprehensive learning method. The latin hypercube sampling based local search is performed every ten iterations. Furthermore, a differential information based mutation operator is employed to maintain the diversity of the swarms. To more explicitly describe the proposed algorithm, the complete flow chat of MA-HPSOL is given in Fig. 7. In the next section, a large number of test problems are used to evaluate the performance of the proposed algorithm. Suppose that the computation cost of one particle in the CLPSO approach is c, the cost of the mutation operator is c_m and the cost of Latin local search is c_l, then the total computation cost of MA-HPSOL for one generation is $(M+1)N(c+c_m)+M(c_l)$. But when solving real-world problems, usually the fitness evaluation accounts for the most time as the PSO is highly computationally efficient. So the algorithm-related computation times are not given in this paper.

4. Experimental study

In this section, we evaluate the performance of MA-HPSOL by solving 16 numerical optimization problems including eight conventional unimodal and multimodal benchmarks,six rotated benchmarks and two composition problems. The test problems are scalable to any number of variables, so we mainly employ the test problems with 10 and 30 variables. We will compare MA-HPSOL with PSO with inertia weight PSOw [5], UPSO [10], FDR-PSO [12], CLPSO [13] and TLPSO [18].

4.1. *Test functions*

In this subsection, we choose 16 function optimization problems to demonstrate the effectiveness of the proposed MA-HPSOL algorithm. They can be classified into four types: unimodal, multimodal, rotated and composite problems. Table 1 tabulates the benchmark test functions with their notable characteristics. The detailed characteristics of these test functions can be found in [32].

4.2. *Sensitivity in relation to parameters*

For the proposed MA-HPSOL algorithm, there are four parameters: M (the number of swarms in the bottom layer), N (the number

```
Set parameters needed in the algorithm;
Initialize the swarms in the bottom layer;
Update the $pbest_{bottom}$ and $gbest_{bottom}$ in each swarm;
Initialize the swarm in the top layer according to $gbest_{bottom}$;
Update the $pbest_{top}$ and $gbest_{top}$;
for $t := 1$ to $Gen$ do
    //learning process of the bottom layer
    Swarms' learning based on Comprehensive Learning;
    Update the $pbest_{bottom}$ and $gbest_{bottom}$ in each swarm;
    Perform mutation operation except for the best particle in each swarm;
    //Information passing from bottom layer to top layer
    Mix $gbest_{bottomi}$ with particles in the top layer($i := 1$ to $M$);
    Select $M$ particles from the mixed groups to form the swarm in the top layer;
    //learning process of the top layer
    Swarm's learning based on Comprehensive Learning;
    Update the $pbest_{top}$ and $gbest_{top}$;
    Perform mutation operation except the best particle;
    //local search
    if Mod$(t, 10) == 0$ then
        Perform Latin Hypercube Sampling based local search;
        Update $gbest_{top}$ if any particle is superior to it;
    end if
    Stop if termination condition is met;
    $t := t + 1$;
end for
```

Fig. 6. The proposed memetic algorithm (MA-HPSOL).

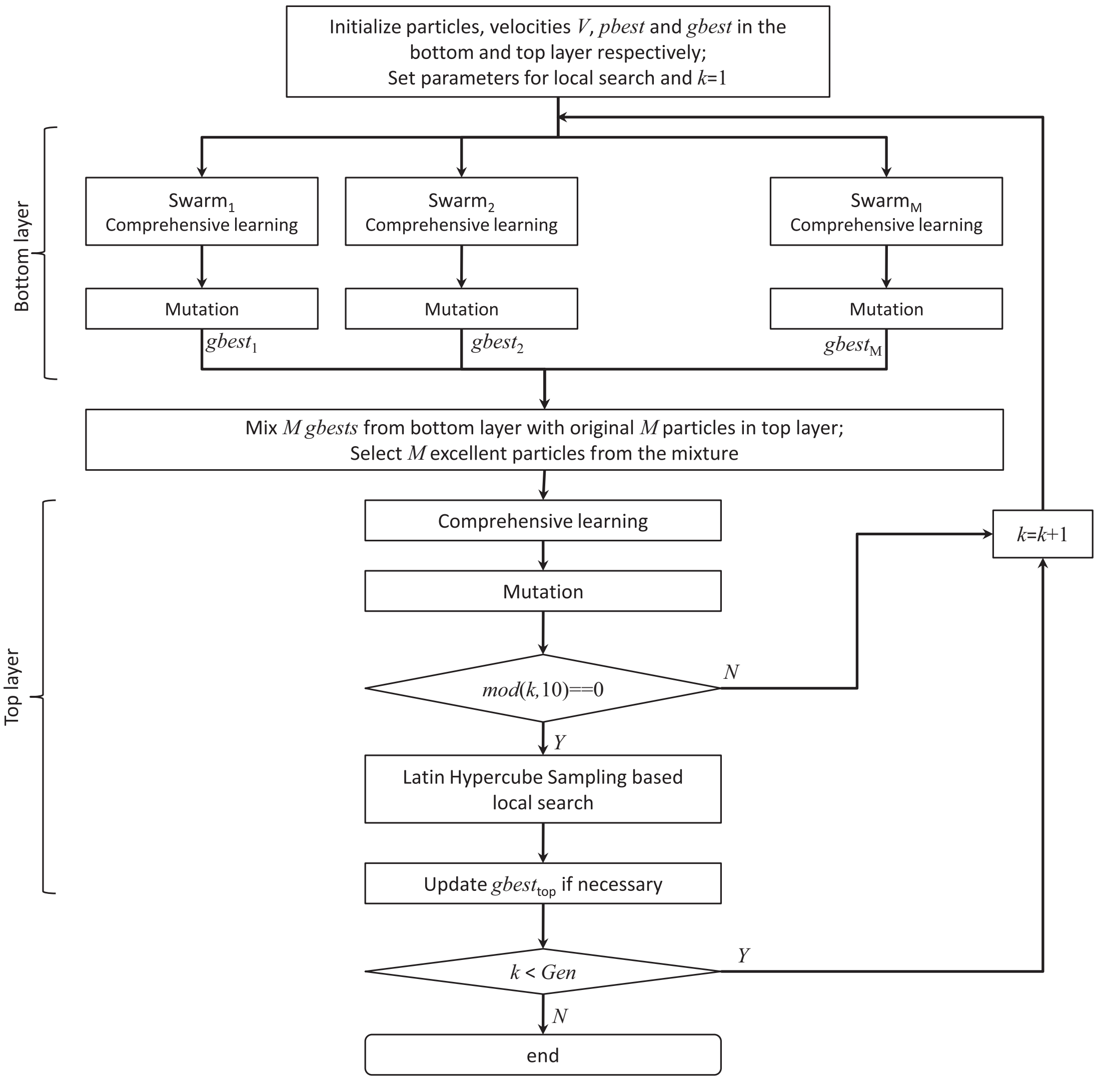
Initialize particles, velocities V, pbest and gbest in the bottom and top layer respectively; Set parameters for local search and k=1
Bottom layer
Swarm1 Comprehensive learning
Swarm2 Comprehensive learning
SwarmM Comprehensive learning
Mutation
gbest1
gbest2
gbestM
Mix M gbests from bottom layer with original M particles in top layer; Select M excellent particles from the mixture
Top layer
Comprehensive learning
Mutation
mod(k,10)==0
N
Y
k=k+1
Latin Hypercube Sampling based local search
Update gbesttop if necessary
k < Gen
Y
N
end

Fig. 7. Flow chat of proposed algorithm (MA-HPSOL).

of particles in each swarm), the sampling scale p and the length of each dimension δ of the hypercube.

Sensitivity in relation to *M* and *N*. The experimental results of MA-HPSOL in optimizing functions 1, 2, 8 and 14 with the number of swarms M increased from 2 to 10 in steps of 1 and the number of particles N in each swarm increased from 2 to 10 in steps of 1. The values of other parameters are as follows: the sampling scale is 10, the length of each dimension δ of the hypercube is twice the length of the corresponding dimension of the selected particle and the mutation probability is the inverse of dimensionality D (here only $D = 10$ is taken into consideration). The maximum number of function evaluations is set at 100,000. The data are statistical average values of the number of function evaluations obtained from 30 independent runs. The results are shown in Fig. 8. From the experimental results of several test functions (functions 1, 2, 8 and 14) depicted in Fig. 8, we can easily find that small values of M and N are encouraged by MA-HPSOL. Therefore, the number of swarms M and the number of particles N in each swarm are both set to 3 in all following experiments.

Sensitivity to the length of each dimension δ of the hypercube. How to get a proper neighborhood for exploitation if a particle has been selected to execute the local search? Because each dimension of the particle may be different from others', the length of each dimension δ of the hypercube should be adaptive to the selected particles. Here, we propose a simple method to specify δ which shows excellent performance in our experiments. The length of each dimension δ of the hypercube is twice the length of the corresponding dimension of the selected particle.

Sensitivity to sampling scale *p*. It is difficult to choose a proper sampling scale p because if we choose a bigger value, the generations for evolution will be few and if we choose a smaller value, the neighborhood of a selected particle may not be exploited sufficiently (if the maximum number of function evaluations is fixed). So we should get a balance between the sampling scale p and the generations of evolution. The experimental results of MA-HPSOL in optimizing functions 8 and 14 with the sampling scale p from 5 to 30 in steps of 5 are shown in Fig. 9. Both the number of swarms M and the number of particles N are set to 3, and other parameters are the same as mentioned above. From the experimental results of functions 8 and 14 depicted in Fig. 9, it is obvious that MA-HPSOL gets better results when the sampling scale is set to 5, 10 and 15. Therefore, the sampling scale p in all following experiments is set to 10.

So all the parameters for MA-HPSOL are shown in Table 2, where δ is set to 2 means that the length of each dimension δ of the hypercube is two times the length of the corresponding dimension of the selected particle and *MAXFES* stands for the maximum number of function evaluations (10,000∗*dimension*).

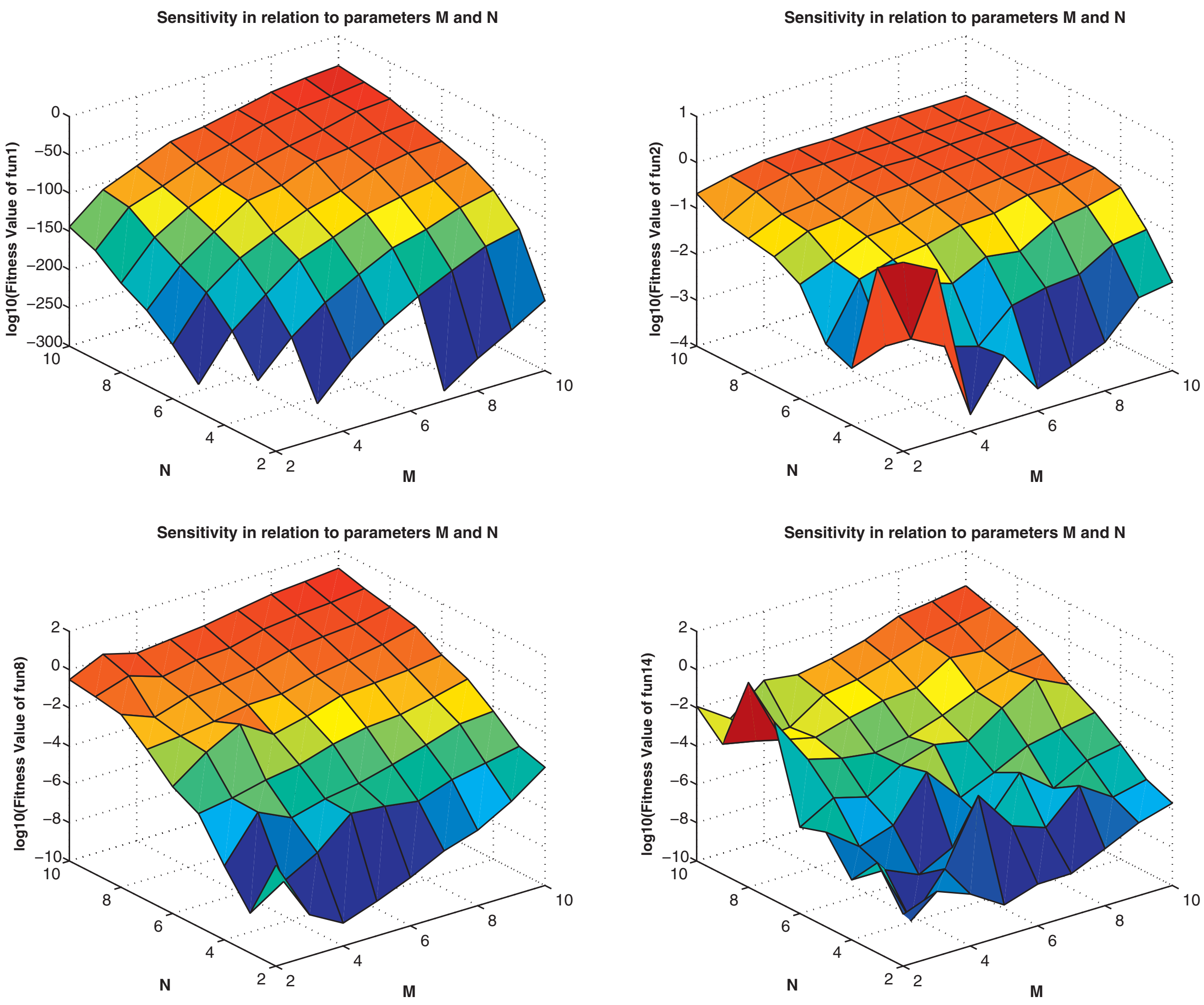

Fig. 8. MA-HPSOL sensitivity in relation to M and N.

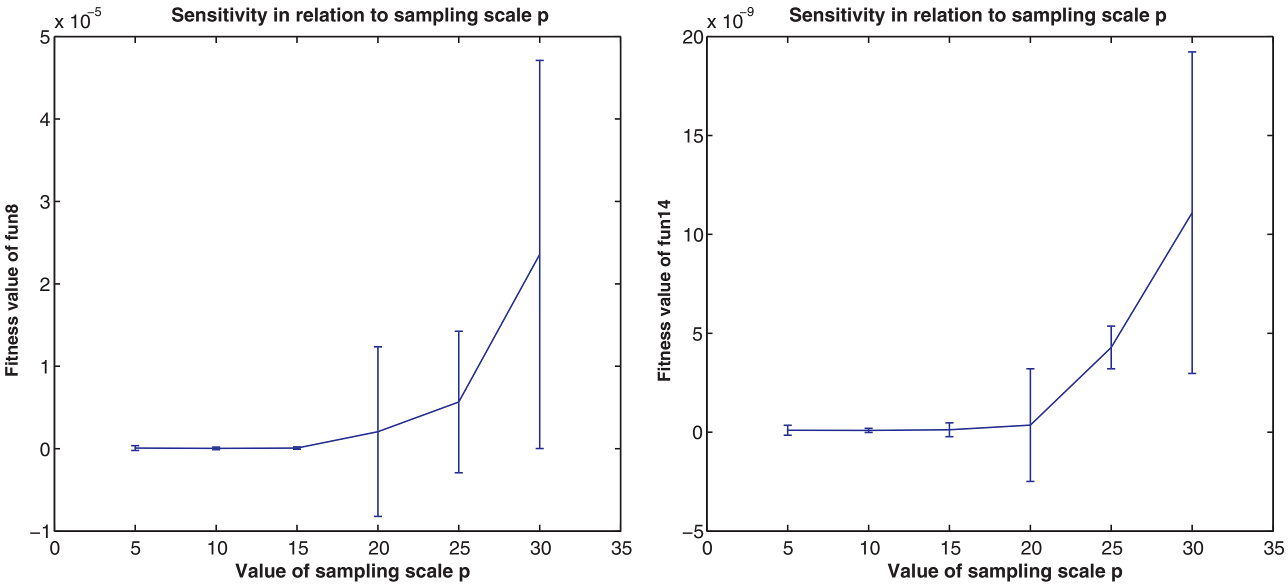

Fig. 9. MA-HPSOL sensitivity in relation to sampling scale *p*.

Table 1
Benchmark functions used in this study.

Function	Range	Characteristics	Optima
$f_1(x)=\sum_{i=1}^{D} x_i^2$	[−100,100]	Unimodal	0
$f_2(x)=\sum_{i=1}^{D-1}(100(x_i^2-x_{i+1})^2+(x_i-1)^2)$	[−2.048,2.048]	Unimodal	0
$f_3(x)=-20\exp\left(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D}x_i^2}\right)$ $-\exp\left(\frac{1}{D}\sum_{i=1}^{D}\cos(2\pi x_i)\right)+20+e$	[−32.768,32.768]	Multimodal	0
$f_4(x)=\sum_{i=1}^{D}\frac{x_i^2}{4000}-\prod_{i=1}^{D}\cos\left(\frac{x_i}{\sqrt{i}}\right)+1$	[−600,600]	Multimodal	0
$f_5(x)=\sum_{i=1}^{D}\{\sum_{k=1}^{kmax}[a^k\cos(2\pi b^k(x_i+0.5))]\}$ $-D\sum_{k=1}^{kmax}\{a^k\cos(2\pi b^k\cdot 0.5)\}$ $a=0.5, b=3, kmax=20;$	[−0.5,0.5]	Multimodal	0
$f_6(x)=\sum_{i=1}^{D}\{x_i^2-10\cos(2\pi x_i)+10\}$	[−5.12,5.12]	Multimodal	0
$f_7(x)=\sum_{i=1}^{D}\{y_i^2-10\cos(2\pi y_i)+10\}$ $y_i=\begin{cases} x_i & \lvert x_i\rvert<0.5 \\ \frac{round(2x_i)}{2} & \lvert x_i\rvert\ge 0.5 \end{cases},$ $i=1,2,\ldots,D$	[−5.12,5.12]	Multimodal	0
$f_8(x)=418.9829D-\sum_{i=1}^{D}\{x_i\sin(\lvert x_i\rvert^{0.5})\}$	[−500,500]	Multimodal	0
$f_9(x)=f_3(y), y=M*x$	[−32.768,32.768]	Rotated	0
$f_{10}(x)=f_4(y), y=M*x$	[−600,600]	Rotated	0
$f_{11}(x)=f_5(y), y=M*x$	[−0.5,0.5]	Rotated	0
$f_{12}(x)=f_6(y), y=M*x$	[−5.12,5.12]	Rotated	0
$f_{13}(x)=f_7(y), y=M*x$	[−5.12,5.12]	Rotated	0
$f_{14}(x)=418.9829D-\sum_{i=1}^{D}z_i$ $z_i=\begin{cases} y_i\sin(\lvert y_i\rvert^{0.5}) & \lvert y_i\rvert\le 500 \\ 0.001(\lvert y_i\rvert-500)^2 & \lvert y_i\rvert>500 \end{cases},$ $i=1,2,\ldots,D; y=M*(x-420.96)+420.96$	[−500,500]	Rotated	0
$f_{15}=CF1$	[−5,5]	Composition	0
$f_{16}=CF2$	[−5,5]	Composition	0

Table 2
Parameters setting for MA-HPSOL.

Parameters	Dimension	
	10*D*	30*D*
M, *N*	{3,3}	{3,3}
p	10	10
δ	2	2
MAXFES	10,000*10	10,000*30

4.3. Experimental results of MA-HPSOL on test functions

In this section, we will give the experimental results obtained by MA-HPSOL in optimizing above-mentioned 16 functions with 10 and 30 variables. Based on the parameter sensitivity analysis, the parameters are set as shown in Table 2. Table 3 shows the statistical results of MA-HPSOL in optimizing the 16 functions with ten variables (10-*D* functions) based on 30 independent runs, which includes the maximum, minimum, mean and standard deviation. The termination criterion in this experiment is to run MA-HPSOL until the number of function evaluations reaches the maximum value 100,000. Obviously MA-HPSOL performs very well on most of the 16 functions. For functions 1, 3, 4, 5, 6, 7, 9, 10, 11, 12 and 13, the maximum, minimum and mean values of the 30 runs are all equal to the optimal values. The performance of MA-HPSOL is stable enough because the diversity is kept on a higher level to avoid premature convergence. But when solving the functions 2, 8, 14, 15 and 16, MA-HPSOL does not get accurate optimal results. Due to the ill-conditional nature of function 2 (Rosenbrock problem) and function 8 (Schwefel problem), which has optimal values at (1,1,. . .,1) and (420.96, 420.96,. . .,420.96), it is hard to adapt quickly to the different optimization stages. Also, function 14 is the rotational version of function 8, so there is some distance between the local optimum found (1.4677e−009) and the global optimum 0.

Table 3
Statistical results of MA-HPSOL in optimizing 10-*D* functions.

Functions	Max	Min	Mean	Std
f_1	0	0	0	0
f_2	1.3825e−003	3.6072e−010	2.3089e−004	3.8665e−004
f_3	0	0	0	0
f_4	0	0	0	0
f_5	0	0	0	0
f_6	0	0	0	0
f_7	0	0	0	0
f_8	2.4559e−007	1.0914e−011	1.1462e−008	4.5219e−008
f_9	0	0	0	0
f_{10}	0	0	0	0
f_{11}	0	0	0	0
f_{12}	0	0	0	0
f_{13}	0	0	0	0
f_{14}	2.3173e−008	2.7285e−012	1.4677e−009	4.3905e−009
f_{15}	2.3587e−009	6.2578e−013	3.0186e−010	2.3079e−010
f_{16}	5.1823e+000	1.8756e+000	3.4610e+000	2.8459e+000

For the two composition functions, most of the solutions are obviously worse than the optimal values. The reason for the poor performance is that both functions are more challenging problems with a randomly located global optimum and several randomly located deep local optima. They are asymmetrical multimodal problems, with different properties in different areas. Due to the complex shape of the composition functions, it is difficult to get the same accurate results as the benchmark functions. However, we find that MA-HPSOL gets relatively good results of these two composite functions when compared with some state-of-the-art algorithms, which will be shown in the following part.

The experimental results for MA-HPSOL in optimizing the 16 functions with 30 variables (30-*D* functions) are shown in Table 4. The maximum number of function evaluations is set at 300,000.

Table 4
Statistical results of MA-HPSOL in optimizing 30-*D* functions.

Functions	Max	Min	Mean	Std
f_1	3.9525e−318	2.9644e−323	1.3562e−319	0
f_2	8.3908e−005	1.1101e−015	5.3122e−006	1.7546e−005
f_3	3.5527e−015	0	1.8948e−015	1.8027e−015
f_4	0	0	0	0
f_5	0	0	0	0
f_6	0	0	0	0
f_7	0	0	0	0
f_8	4.7294e−011	0	9.2162e−012	9.1629e−012
f_9	0	0	0	0
f_{10}	0	0	0	0
f_{11}	0	0	0	0
f_{12}	0	0	0	0
f_{13}	0	0	0	0
f_{14}	2.5466e−011	0	5.0932e−012	5.7601e−012
f_{15}	4.2483e−016	8.4579e−018	8.9180e−017	4.6175e−017
f_{16}	1.2781e+001	1.2472e+000	4.5891e+000	3.4578e+000

The other parameters are the same as those for 10-*D* functions. The statistical results in Table 4 are obtained from 30 independent runs. As shown in Table 4, when solving the functions 4, 5, 6, 7, 9, 10, 11, 12 and 13, the statistical results including the maximum, minimum and mean values for the 30 runs are all equal to the optimal values. The results of functions 1 and 3 are not so good as the results obtained in the 10-*D* experiment. The main reason accounting for this phenomenon may be the lack of a sufficient number of particles for exploring the feasible space. We use the same pair of $\{M,N\}=\{3,3\}$ for both 10-*D* and 30-*D* experiments, which means the total particles in the population is 9. This number of particles is proper for 10-*D* experiments, and MA-HPSOL obtains promising results as do some other algorithms [13]. But the landscape of test 30-*D* functions is so complex that it is difficult to explore such a high dimensional feasible space ($D=30$) with so few particles. As the results of functions 2, 8, 14, 15 and 16 show, MA-HPSOL can get values which are very close to the optima. Also, the standard are very small for all of these functions, which means that MA-HPSOL exhibits excellent stability over all 30 runs.

4.4. Comparisons with state-of-the-art algorithms

In order to further verify the effectiveness of MA-HPSOL, we use experiments to evaluate the performance of MA-HPSOL by comparing it with five existing algorithms, PSOw [5], UPSO [10], FDR-PSO [12], CLPSO [13] and TLPSO [18]. For easy comparison with state-of-the-art algorithms, the population size for all six algorithms is set to 9. Any specific parameters are set exactly the same as in the original work. The termination criteria is to run the algorithms until the number of function evaluations reaches the maximum value 100,000. All the results given in Table 5 are based on 30 independent runs.

Table 5
Results of six algorithms for 10-*D* results.

PSOs	Func			
	f_1	f_2	f_3	f_4
PSOw	1.1911e−222 ± 0.0000e−000	6.7300e−001±1.2280e−000	2.3566e−014 ± 4.3135e−014	8.9226e−002 ± 4.1675e−002
UPSO	0 ± 0	9.1158e−001 ± 1.6832e−000	1.7941e+000 ± 1.8726e+000	1.0769e−001 ± 1.2112e−001
FDR-PSO	2.3711e−292 ± 0.0000e−000	5.3167e−001 ± 1.3785e−000	1.1842e−014 ± 9.2046e−015	1.1020e−001 ± 4.7358e−002
CLPSO	1.0661e−121 ± 5.8364e−121	2.4131e+000 ± 1.7242e+000	3.5527e−015 ± 0.0000e−000	2.5449e−003 ± 5.2275e−003
TLPSO	7.4796e−149 ± 4.0967e−148	5.1611e+000 ± 1.2933e+000	3.5290e−014 ± 6.8554e−014	2.0856e−002 ± 2.7759e−002
MA-HPSOL	0 ± 0	2.3089e−004 ± 3.8665e−004	0 ± 0	0 ± 0

PSOs	Func			
	f_5	f_6	f_7	f_8
PSOw	8.0020e−004 ± 1.7287e−003	5.3396e+000 ± 3.1839e+000	3.4667e+000 ± 2.0965e+000	5.4087e+002 ± 2.0281e+002
UPSO	1.1946e−000 ± 8.3523e−001	1.1613e+001 ± 6.8241e+000	1.0333e+000 ± 1.2389e+000	9.9245e+002 ± 2.8162e+002
FDR-PSO	2.5278e−003 ± 1.0585e−002	3.7145e+000 ± 2.7396e+000	1.0000e+000 ± 1.5313e+000	6.9418e+002 ± 1.9950e+002
CLPSO	0 ± 0	0 ± 0	0 ± 0	2.4358e−005 ± 8.5697e−005
TLPSO	2.2998e−013 ± 1.2503e−012	1.7962e+000 ± 4.8035e+000	1.6667e−001 ± 9.1287e−001	1.5193e+003 ± 3.2714e+002
MA-HPSOL	0 ± 0	0 ± 0	0 ± 0	1.1462e−008 ± 4.5219e−008

PSOs	Func			
	f_9	f_{10}	f_{11}	f_{12}
PSOw	2.0899e−001 ± 4.8204e−001	1.4705e−001 ± 7.8549e−002	5.5281e−001 ± 7.0310e−001	9.4189e+000 ± 4.1126e+000
UPSO	1.3929e+000 ± 1.2389e+000	1.0107e−001 ± 7.2331e−002	2.3732e+000 ± 1.2709e+000	1.5531e+001 ± 5.8409e+000
FDR-PSO	1.5402e−001 ± 3.99.9e−001	1.6642e−001 ± 5.6494e−002	3.4951e−001 ± 4.3231e−001	1.1077e+001 ± 4.9969e+000
CLPSO	5.7048e−006 ± 1.4831e−005	2.5077e−004 ± 1.3499e−003	5.2010e−004 ± 2.8571e−004	4.1516e+000 ± 1.8617e+000
TLPSO	1.3536e−013 ± 5.9653e−013	3.5548e−002 ± 4.6763e−002	9.0035e−001 ± 1.0941e−000	1.7993e+001 ± 6.5542e+000
MA-HPSOL	0 ± 0	0 ± 0	0 ± 0	0 ± 0

PSOs	Func			
	f_{13}	f_{14}	f_{15}	f_{16}
PSOw	8.7333e+000 ± 2.6773e+000	7.9385e+002 ± 3.2423e+002	1.2333e+002 ± 1.3047e+002	1.5285e+002 ± 2.1481e+002
UPSO	1.3171e+001 ± 6.4240e+000	1.0933e+003 ± 3.9505e+002	6.6667e+001 ± 7.5810e+001	1.3183e+002 ± 1.4388e+002
FDR-PSO	1.0267e+001 ± 3.6192e+000	1.0588e+003 ± 3.7468e+002	1.1333e+002 ± 1.1958e+002	1.4412e+002 ± 1.9202e+002
CLPSO	3.9943e+000 ± 1.1333e+000	2.3378e+002 ± 1.8373e+002	9.4634e+000 ± 2.8594e+001	4.9802e+000 ± 2.5518e+000
TLPSO	1.0502e+001 ± 5.3488e+000	1.5355e+003 ± 3.8439e+002	5.3404e+001 ± 8.5992e+001	6.6410e+001 ± 1.0810e+002
MA-HPSOL	0 ± 0	1.4677e−009 ± 4.3905e−009	3.0186e−010 ± 2.3079e−009	3.4610e+000 ± 2.8459e+000

Table 6
Results of six algorithms for 30-*D* results.

PSOs	Func			
	f_1	f_2	f_3	f_4
PSOw	5.4003e−014 ± 2.7946e−013	2.0220e+001 ± 2.7331e+000	1.2901e−000 ± 7.9643e−001	3.0255e−002 ± 3.0136e−002
UPSO	5.7543e−048 ± 1.8637e−047	1.7619e+001 ± 2.8343e+000	9.3904e+000 ± 3.2206e+000	4.6661e−001 ± 1.0997e−000
FDR-PSO	4.1352e−037 ± 1.3299e−036	1.8431e+001 ± 2.0006e+000	1.9836e−001 ± 4.6920e−001	2.0955e−002 ± 2.5570e−002
CLPSO	4.2811e−100 ± 2.3448e−099	1.9102e+001 ± 5.9737e+000	1.0184e−014 ± 3.7007e−015	0 ± 0
TLPSO	5.5844e−000 ± 1.7482e−000	3.4220e+001 ± 2.3256e+001	4.3912e+000 ± 6.7563e+000	3.7541e+000 ± 1.3289e+001
MA-HPSOL	1.3562e−319 ± 0	5.3122e−006 ± 1.7546e−005	1.8948e−015 ± 1.8027e−015	0 ± 0

PSOs	Func			
	f_5	f_6	f_7	f_8
PSOw	2.4549e+000 ± 1.7194e+000	5.4391e+001 ± 1.8495e+001	4.7067e+001 ± 1.0748e+001	3.3460e+003 ± 5.3609e+002
UPSO	1.7658e+001 ± 3.7136e+000	8.8207e+001 ± 2.4246e+001	6.8608e+001 ± 3.6082e+001	5.1012e+003 ± 8.0621e+002
FDR-PSO	1.4589e+000 ± 1.3015e+000	4.8653e+001 ± 7.9526e+000	2.0467e+001 ± 1.1834e+001	3.8710e+003 ± 4.6738e+002
CLPSO	0 ± 0	9.2863e−001 ± 8.2351e−001	3.5333e+000 ± 1.8144e+000	3.2373e+002 ± 1.7017e+002
TLPSO	8.6334e+000 ± 6.2190e+000	1.2334e+002 ± 3.7287e+001	9.3403e+001 ± 3.0241e+001	6.9289e+003 ± 7.3891e+002
MA-HPSOL	0 ± 0	0 ± 0	0 ± 0	9.2162e−012 ± 9.1629e−012

PSOs	Func			
	f_9	f_{10}	f_{11}	f_{12}
PSOw	2.3447e−000 ± 7.2235e−001	2.5026e−002 ± 2.4675e−002	8.3977e+000 ± 2.8246e+000	6.9846e+001 ± 1.8030e+001
UPSO	9.9534e+000 ± 2.2655e+000	1.2333e−001 ± 2.4671e−001	2.3634e+001 ± 2.7319e+000	1.0299e+002 ± 2.9672e+001
FDR-PSO	1.7451e+000 ± 5.9430e−001	1.9906e−002 ± 2.4336e−002	6.0272e+000 ± 1.7910e+000	6.0891e+001 ± 1.3842e+001
CLPSO	2.1219e−005 ± 1.0226e−004	2.4710e−004 ± 1.3502e−003	3.0519e+000 ± 2.0973e+000	3.9856e+001 ± 8.7627e+000
TLPSO	4.7013e+000 ± 7.3372e+000	2.9873e−002 ± 1.2639e−001	1.6220e+001 ± 7.8068e+000	1.3424e+002 ± 3.1714e+001
MA-HPSOL	0 ± 0	0 ± 0	0 ± 0	0 ± 0

PSOs	Func			
	f_{13}	f_{14}	f_{15}	f_{16}
PSOw	7.9900e+001 ± 2.2287e+001	4.4924e+003 ± 7.9191e+002	3.0000e+001 ± 7.0221e+001	4.2018e+001 ± 8.9057e+001
UPSO	1.1684e+002 ± 2.7428e+001	5.9841e+003 ± 8.5677e+002	9.3333e+001 ± 1.4606e+002	1.4799e+002 ± 1.0022e+002
FDR-PSO	6.3675e+001 ± 1.6926e+001	4.6024e+003 ± 8.3439e+002	3.6667e+001 ± 7.6489e+001	6.4385e+001 ± 1.3767e+002
CLPSO	3.8088e+001 ± 8.5958e+000	2.8051e+003 ± 5.7583e+002	4.3520e−003 ± 2.3791e−002	2.1560e+001 ± 7.2032e+001
TLPSO	1.1142e+002 ± 3.3820e+001	7.7998e+003 ± 7.3494e+002	5.4619e+001 ± 3.5070e+001	3.9283e+002 ± 3.5491e+002
MA-HPSOL	0 ± 0	5.0932e−012 ± 5.7601e−012	8.9180e−017 ± 4.6175e−017	4.5891e+000 ± 3.4578e+000

Furthermore, a distance function *Index*(*D*) for describing the mean distance between the optimal solution and the obtained best solution is defined as follows [33].

$$Index(D) = \frac{|f_{opt}(D) - f_{best}(D)|}{D}. \qquad (6)$$

where $f_{opt}(D)$ and $f_{best}(D)$ are the optimal solution and the obtained best solution, respectively. This metric is usually used to compare the decreasing velocities of the differences between the solutions obtained by all kinds of evolutionary algorithms and the target solution. In this paper, the optima for all the test functions are 0 and the obtained best solutions are usually very close to 0, so we use the $log10$ ($f_{best}(D)$) instead of $f_{best}(D)$ for narrowing the interval of metric. But the *abs*(*log*) function is not monotonic so we modify the *Index*(*D*) to *Dist*(*D*) as follows so that we can easily visualize the results of each algorithm.

$$Dist(D) = \frac{(log10(f_{best}(D)) - f_{opt}(D))}{D} \qquad (7)$$

Fig. 10 presents the *Dist*(*D*) values in terms of the best fitness value of the median run of each algorithm for each test function ($D = 10$). We record the best solutions every 5000 function evaluations for each test problem with total function evaluations 100,000. So the interval for the horizontal coordinate is [1,20] and the vertical coordinate shows the *Dist*(*D*).

From the results in Table 5 and Fig. 10, we observe that MA-HPSOL surpasses all other algorithms on all functions except function 8. The performance of CLPSO in optimizing function 8 is superior to MA-HPSOL. However, when we run both CLPSO and MA-HPSOL 50 independent times we find that CLPSO holds a very small probability (3/50) to trap in local optima 118.4383 and 236.8767 while MA-HPSOL still obtain results with precision 10^{-8}. The convergence characteristic of MA-HPSOL is very promising in optimizing unimodal, multimodal, rotated multimodal and composite problems. For UPSO, it just performs well on unimodal function 1 and also other algorithms get good results. PSOw shows good convergence characteristics on functions 1, 3 and 9 while FDR-PSO has a relatively good convergence for functions 2, 3 and 9. This is a reasonable phenomenon because function 9 is just the rotational version of function 3. TLPSO shows good convergence in optimizing functions 1, 3, 5, 6, 7, 9 and 10. Because particles in MA-HPSOL are trained with the comprehensive learning method, MA-HPSOL and CLPSO share some similar convergence characteristics in optimizing functions 1, 3, 4, 8, 9, 15 and 16 with the difference that MA-HPSOL converges faster than CLPSO. Especially for functions 3, 4, 5, 6, 7, 9, 10, 11, 12 and 13, MA-HPSOL converges to the global optimum in less than 20,000 function evaluations on the whole. This is mainly caused by multi-swarms in the hierarchical architecture and the local search strategy.

Table 6 gives the means and standard deviations of the 30 runs of the six algorithms on the sixteen test functions with $D = 30$. As the convergence graphs are similar to the 10-*D* problems, they are not presented here. It is a acid test for these algorithms holding a population with just 9 particles. From the results in Table 6, we

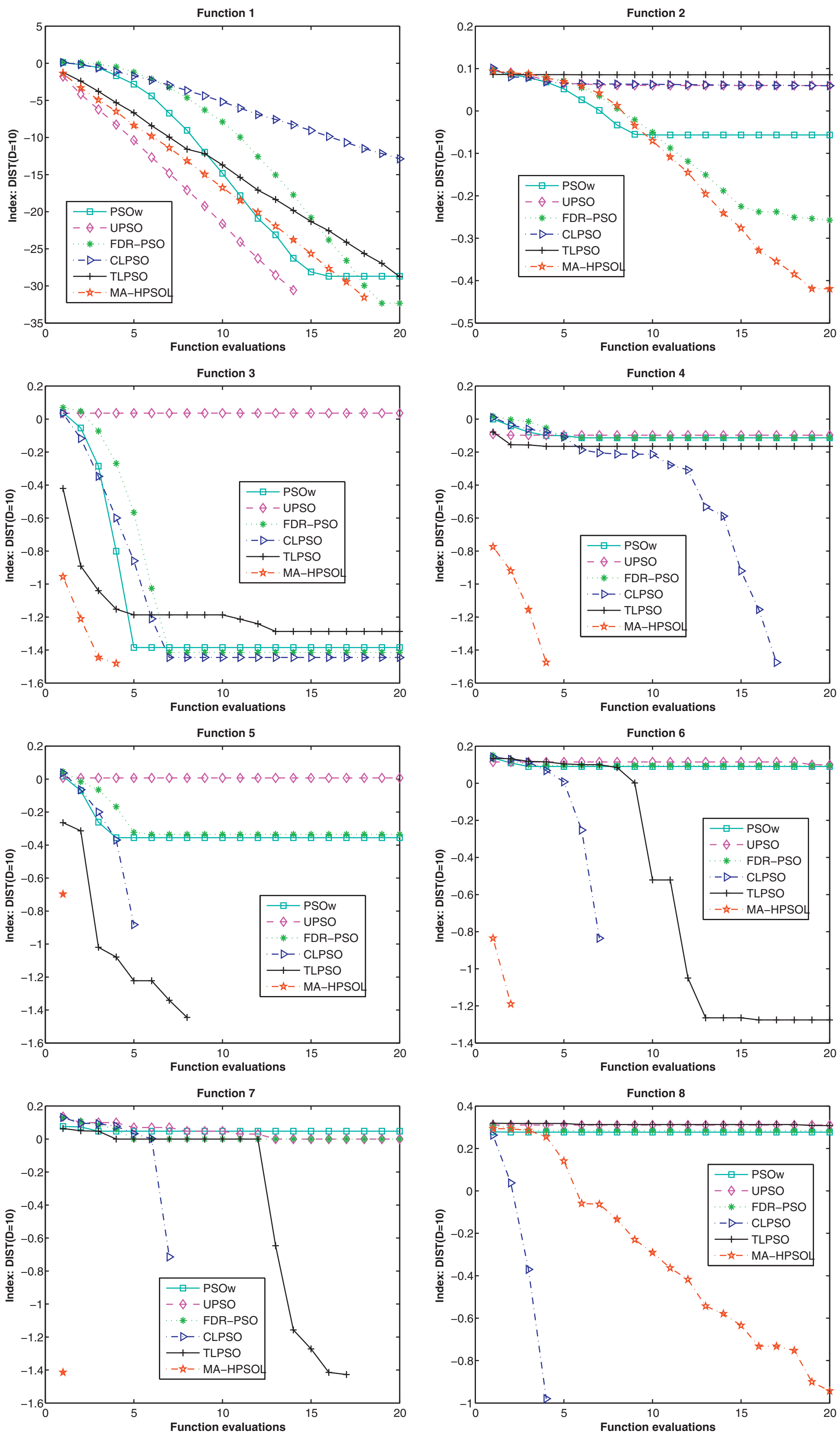

Fig. 10. The median *dist*(*D*) values of 10-*D* test functions.

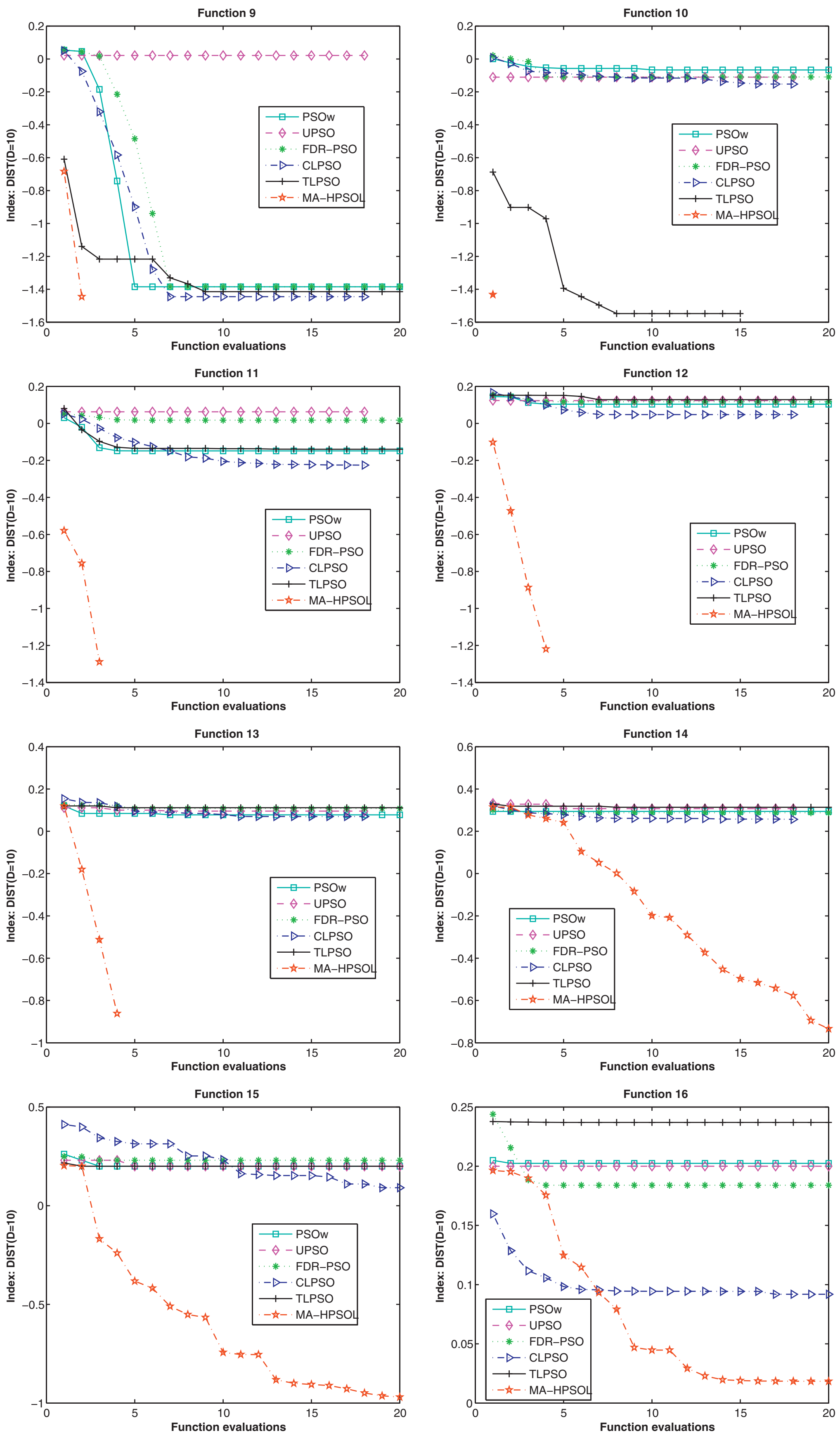

Fig. 10. (*continued*)

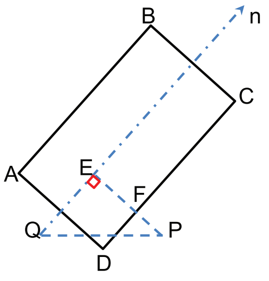

Fig. 11. Definition of cylindricity error.

can observe that the performance of almost all algorithms except MA-HPSOL degrade dramatically in optimizing high-dimensional problems with a small population size. Taking CLPSO for example, it can attain the precision of 10^{-12} with population size 40, but it only 10^3 with population size 9.

4.5. *Cylindricity error evaluation based on MA-HPSOL*

In the past few years, many kinds of evolutionary algorithms have contributed to optimize a wide range of manufacturing process [34–36], whose demands to be more robust, more flexible, more complex are ever increasing. Cylindrical features have become one of the most important features in mechanical designs. They contribute significantly to fundamental mechanical products such as transmission systems, revolving devices and injection molds, to achieve the intended functionalities. Therefore, evaluating cylindricity error precisely is very important in high precision manufacturing. Many attempts have been made for evaluating the cylindricity error [37,38].

The definition of cylindricity error can be stated as follows [39]. Fig. 11 illustrates the cross section of a cylinder with axis direction $\mathbf{n}(l,m,1)$ and radius R. The projection of a measured point P onto the cylinder is F. Assuming the axis passes the point $Q(x_0, y_0, 0)$, then the axis function can be expressed as $(x-x_0)/l=(y-y_0)/m=z$. The distance from $P_i(i=1, 2, \ldots, N)$ to the axis is

$$e_i = |EP_i| = \frac{|\vec{QP_i} \times \mathbf{n}|}{|\mathbf{n}|} = \frac{\begin{vmatrix} \vec{i} & \vec{j} & \vec{k} \\ x_i - x_0 & y_i - y_0 & z_i \\ l & m & 1 \end{vmatrix}}{\sqrt{l^2+m^2+1}}, \quad (8)$$

where $|\cdot|$ means the length of a vector in the Euclidean space. Mathematically, the cylindricity error evaluation can be formulated as an optimization problem with parameter vector (x_0, y_0, l, m). Hence, the fitness function of evaluating cylindricity error under minimum zone cylinder (MZC) criterion is aiming at minimizing the objective function:

$$f(x_0, y_0, l, m) = \max(e_i) - \min(e_i). \quad (9)$$

Here we will evaluate the cylindricity error by the above proposed MA-HPSOL algorithm and related parameters are set as follows: (1) The MA-HPSOL dependent parameters are set as shown in Table 2; (2) Dimension of particles is 4, which is the length of parameter vector (x_0, y_0, l, m); (3) Terminal condition: maximum generations 100. The remaining parameters are the same as [13]. The measurement data sets are introduced from Refs. [38,40]. All parameters are initialized in [−1,1]. The evaluating results are given in Tables 7 and 8. As shown in Tables 7 and 8, the proposed MA-HPSOL algorithm is a competitive approach in cylindricity error evaluation, which is obviously a complicated optimization problem. When comparing with other types of evolutionary algorithms (Improved GA [37,41], PSO [38], PSO-DE [39]), the results obtained by MA-HPSOL are better than that listed in existing literatures.

Table 7
Results of data set 1 cylindricity error evaluation.

Parameter	Improved GA [37]	PSO [38]	MA-HPSOL
x_0	0.0009250	0.003315	0.0020284
y_0	−0.0002253	0.002814	0.0000496
z_0	0.0014643	0	0
l	0.0000435	−0.00052	0.0000591
m	0.0000162	0.000609	0.0000214
n	0.9996235	1	1
Cylindricity	0.0105976	0.025368	0.0104864

Table 8
Results of data set 2 cylindricity error evaluation.

Parameter	Improved GA [41]	PSO-DE [39]	MA-HPSOL
x_0	0.011853	0.010650	0.0106429
y_0	0.047689	0.046918	0.0469181
z_0	0	0	0
l	−0.000674	−0.000619	−0.000619
m	0.002960	−0.002915	−0.002915
n	1	1	1
Cylindricity	0.184274	0.18397196	0.1839592

5. Conclusion and future work

This paper presents a high performance memetic algorithm (MA-HPSOL) to deal with complex numerical optimization problems. Within the framework of the proposed algorithm, there are three main components: an hierarchical particle swarm optimizer for exploration, a local search method based on latin hypercube sampling for exploitation and a mutation operator using differential information.

Concretely, the hierarchical PSO is composed of two layers: the bottom layer and the top layer. Particles in each swarm of the bottom layer evolve independently, which means each swarm is a niche with no influence on other swarms. Global best position in each swarm of the bottom layer becomes the candidate of the particle in the top layer, so the global best position in the swarm of the top layer steers the particles in each swarm of the bottom layer indirectly. The local search strategy, latin hypercube sampling, aims at exploiting the best solutions found so far uniformly. Both such exploration and the exploitation operators can help keep the diversity of whole population on a higher level to avoid particles' trapping into local optima. Even if particles in one swarm are trapped in local optima, other swarms are also likely to reach the global optima. Furthermore, a mutation operator, aiming at modifying the particles' positions based on differential information, is used. According to the experimental results on 16 functions, the proposed memetic algorithm (MA-HPSOL) has excellent performance to find global optimal solutions. MA-HPSOL is used to evaluate the cylindricity error and the experimental results show that it can obtain competitive performance as well.

For our future work, two aspects, quantitatively depicting the diversity of the whole population and imposing mutual communication among swarms in the bottom layer, will be investigated in depth.

Acknowledgments

This work was partially supported by the National Basic Research Program of China (grant no. 2009CB320901) and the European Union Seventh Framework Program (grant no. 247619). We would like to thank Prof. P.N. Suganthan for providing the source code of "CLPSO".

References

[1] H. Azamathulla, F. Wu, Support vector machine approach for longitudinal dispersion coefficients in natural streams, Applied Soft Computing 11 (2) (2011) 2902–2905.
[2] H. Azamathulla, A. Ghani, C. Chang, Z. Hasan, N. Zakaria, Machine learning approach to predict sediment load—a case study, Clean-Soil, Air, Water 38 (10) (2010) 969–976.
[3] J. Kennedy, R. Eberhart, Particle swarm optimization, in: Proceedings of IEEE International Conference on Neural Networks, vol. 4, 1995, pp. 1942–1948.
[4] R. Eberhart, J. Kennedy, A new optimizer using particle swarm theory, in: Proceedings of the Sixth International Symposium on Micro Machine and Human Science, 1995, pp. 39–43.
[5] Y. Shi, R. Eberhart, A modified particle swarm optimizer, in: Proceedings of IEEE International Conference on Evolutionary Computation, 1998, pp. 69–73.
[6] J. Kennedy, Small worlds and mega-minds: effects of neighborhood topology on particle swarm performance, in: Proceedings of IEEE Congress on Evolutionary Computation, vol. 3, 1999.
[7] J. Kennedy, R. Mendes, Population structure and particle swarm performance, in: Proceedings of IEEE Congress on Evolutionary Computation, vol. 2, IEEE, 2002, pp. 1671–1676.
[8] P. Suganthan, Particle swarm optimiser with neighbourhood operator, in: Proceedings of IEEE Congress on Evolutionary Computation, vol. 3, 1999.
[9] W. Jian, Y. Xue, J. Qian, Improved particle swarm optimization algorithms study based on the neighborhoods topologies, in: Proceedings of IEEE Annual Conference of Industrial Electronics Society, vol. 3, 2004, pp. 2192–2196.
[10] K. Parsopoulos, M. Vrahatis, UPSO: a unified particle swarm optimization scheme, Lecture Series on Computer and Computational Sciences 1 (2004) 868–873.
[11] R. Mendes, J. Kennedy, J. Neves, The fully informed particle swarm: simpler, maybe better, IEEE Transactions on Evolutionary Computation 8 (3) (2004) 204–210.
[12] T. Peram, K. Veeramachaneni, C. Mohan, Fitness–distance-ratio based particle swarm optimization, in: Proceedings of IEEE Symposium on Swarm Intelligence, 2003, pp. 174–181.
[13] J. Liang, A. Qin, P. Suganthan, S. Baskar, Comprehensive learning particle swarm optimizer for global optimization of multimodal functions, IEEE Transactions on Evolutionary Computation 10 (3) (2006) 281–295.
[14] Y. Jiang, T. Hu, C. Huang, X. Wu, An improved particle swarm optimization algorithm, Applied Mathematics and Computation 193 (1) (2007) 231–239.
[15] S. Yang, C. Li, A clustering particle swarm optimizer for locating and tracking multiple optima in dynamic environments, IEEE Transactions on Evolutionary Computation 14 (6) (2010) 959–974.
[16] H. Wang, S. Yang, W. Ip, D. Wang, A particle swarm optimization based memetic algorithm for dynamic optimization problems, Natural Computing 9 (3) (2010) 703–725.
[17] H. Wang, S. Yang, W.H. Ip, D. Wang, A memetic particle swarm optimisation algorithm for dynamic multi-modal optimisation problems, International Journal of Systems Science 43 (7) (2012) 1268–1283.
[18] C. Chen, Two-layer particle swarm optimization for unconstrained optimization problems, Applied soft computing 11 (1) (2011) 295–304.
[19] P. Angeline, Using selection to improve particle swarm optimization, in: Proceedings of IEEE International Conference on Evolutionary Computation, 1998, pp. 84–89.
[20] M. Lovbjerg, T. Rasmussen, T. Krink, Hybrid particle swarm optimiser with breeding and subpopulations, in: Proceedings of the Third Genetic and Evolutionary Computation Conference, vol. 1, Citeseer, 2001, pp. 469–476.
[21] V. Miranda, N. Fonseca, EPSO-evolutionary particle swarm optimization, a new algorithm with applications in power systems, in: Transmission and Distribution Conference and Exhibition: Asia Pacific. IEEE/PES, vol. 2, 2002, pp. 745–750.
[22] K. Parsopoulos, M. Vrahatis, On the computation of all global minimizers through particle swarm optimization, IEEE Transactions on Evolutionary Computation 8 (3) (2004) 211–224.
[23] F. Van den Bergh, A. Engelbrecht, A cooperative approach to particle swarm optimization, IEEE Transactions on Evolutionary Computation 8 (3) (2004) 225–239.
[24] S. Ling, H. Iu, K. Chan, H. Lam, B. Yeung, F. Leung, Hybrid particle swarm optimization with wavelet mutation and its industrial applications, IEEE Transactions on Systems, Man, and Cybernetics, Part B 38 (3) (2008) 743–763.
[25] X. Zhao, A perturbed particle swarm algorithm for numerical optimization, Applied Soft Computing 10 (1) (2010) 119–124.
[26] H. Gao, W. Xu, Particle swarm algorithm with hybrid mutation strategy, Applied Soft Computing 11 (8) (2011) 5129–5142.
[27] S. Hsieh, T. Sun, C. Liu, S. Tsai, Efficient population utilization strategy for particle swarm optimizer, IEEE Transactions on Systems, Man, and Cybernetics: Part B 39 (2) (2009) 444–456.
[28] R. Brits, A. Engelbrecht, F. Van den Bergh, Solving systems of unconstrained equations using particle swarm optimization, in: Proceedings of IEEE International Conference on Systems, Man and Cybernetics, vol. 3, 2002, pp. 102–107.
[29] N. Huy, O. Soon, L. Hiot, N. Krasnogor, Adaptive cellular memetic algorithms, Evolutionary Computation 17 (2) (2009) 231–256.
[30] A. Qin, V. Huang, P. Suganthan, Differential evolution algorithm with strategy adaptation for global numerical optimization, IEEE Transactions on Evolutionary Computation 13 (2) (2009) 398–417.
[31] M. McKay, R. Beckman, W. Conover, A comparison of three methods for selecting values of input variables in the analysis of output from a computer code, Technometrics (1979) 239–245.
[32] J. Liang, P. Suganthan, K. Deb, Novel composition test functions for numerical global optimization, in: Proceedings of IEEE Symposium on Swarm Intelligence, 2005, pp. 68–75.
[33] S. Ho, L. Shu, J. Chen, Intelligent evolutionary algorithms for large parameter optimization problems, IEEE Transactions on Evolutionary Computation 8 (6) (2004) 522–541.
[34] K. Chan, C. Kwong, Y. Tsim, A genetic programming based fuzzy regression approach to modelling manufacturing processes, International Journal of Production Research 48 (7) (2010) 1967–1982.
[35] K. Chan, C. Kwong, Y. Tsim, Modelling and optimization of fluid dispensing for electronic packaging using neural fuzzy networks and genetic algorithms, Engineering Applications of Artificial Intelligence 23 (1) (2010) 18–26.
[36] K. Chan, C. Kwong, H. Jiang, M. Aydin, T. Fogarty, A new orthogonal array based crossover, with analysis of gene interactions, for evolutionary algorithms and its application to car door design, Expert Systems with Applications 37 (5) (2010) 3853–3862.
[37] H. Lin, Y. Peng, Evaluation of cylindricity error based on an improved GA with uniform initial population, in: Proceedings of IITA International Conference on Control, Automation and Systems Engineering, IEEE, 2009, pp. 311–314.
[38] J. Mao, Y. Cao, J. Yang, Implementation uncertainty evaluation of cylindricity errors based on geometrical product specification (GPS), Measurement 42 (5) (2009) 742–747.
[39] X. Zhang, X. Jiang, P. Scott, A reliable method of minimum zone evaluation of cylindricity and conicity from coordinate measurement data, Precision Engineering 35 (3) (2011) 484–489.
[40] K. Carr, P. Ferreira, Verification of form tolerances part II: cylindricity and straightness of a median line, Precision Engineering 17 (2) (1995) 144–156.
[41] X. Wen, A. Song, An improved genetic algorithm for planar and spatial straightness error evaluation, International Journal of Machine Tools and Manufacture 43 (11) (2003) 1157–1162.