Research Journal of Applied Sciences, Engineering and Technology 5(8): 2576-2585, 2013
DOI:10.19026/rjaset.5.4699
ISSN: 2040-7459; e-ISSN: 2040-7467

Submitted: August 07, 2012 Accepted: September 24, 2012 Published: March 15, 2013

Research Article

Parallel Multi-Swarm PSO Based on K-Medoids and Uniform Design

[1, 2]Jie Zhang, [1]Yuping Wang and [2]Junhong Feng
[1]School of Computer Science and Technology, Xidian University, Xi'an 710071, P.R. China
[2]Department of Computer Science and Technology, Guangzhou University Sontan College, Zengcheng 511370, Guangzhou, P.R. China

Abstract: PAM (Partitioning around Medoid) is introduced to divide the swarm into several different sub-populations. PAM is one of *k*-medoids clustering algorithms based on partitioning methods. It attempts to divide *n* objects into *k* partitions. This algorithm overcomes the drawbacks of being sensitive to the initial partitions in *k*-means algorithm. In the parallel PSO algorithms, the swarm needs to be divided into several different smaller swarms. This study can be excellently completed by PAM. The aim of clustering is that particles within the same sub-population are relative concentrative, so that they can be relatively easy to learn. The purposes of this strategy are that the limited time will be spent on the most effective search; therefore, the search efficiency can also be significantly improved. In order to explore the whole solution space evenly, uniform design is introduced to generate an initial population, in which the population members are scattered uniformly over the feasible solution space. In evolution, uniform design is also introduced to replace some worse individuals. Based on abovementioned these technologies, a novel algorithm, parallel multi-swarm PSO based on k-medoids and uniform design, is proposed. A difference between the proposed algorithm and the others is that PAM and uniform design are both firstly introduced to parallel PSO algorithms.

Keywords: Multi-swarm, medoids, PAM, parallel, particle swarm optimization, uniform design

INTRODUCTION

The particle swarm Optimization Algorithm (PSO) is one of swarm intelligence optimization algorithms, which were proposed by Eberhart and Kennedy (1995), and Kennedy and Eberhart (1995). The idea is originated from the exchange and sharing of information among bird individuals in the process of searching food. Each individual can benefit from discovery and flight experience of the others. In PSO algorithms, the particle swarm is initialized randomly in feasible solution space and each particle has initial speed and position. The track is updated through individual best position and the global best location, which is found by the entire population. This makes particles constantly move to the optimal solution and ultimately move to the global optimal solution.

PSO algorithm is easy to fall into local optimization, to cause premature convergence problem. This is mainly because the proportion of tracking global best position is too much. The accumulating of all particles' tracking leads to track much faster. Therefore, population diversity will have a rapid decline. This causes large amounts of particles to turn into similar particle, even only a few particles or one particle. Namely, the main reason is rapid decline of population diversity. Thus, the iterations curve decline a great deal more quickly in the beginning, but much more slowly in the later. This is particularly obvious in the vicinity of the optimal solution.

Parallel algorithm (Dudy *et al*., 2007; Antonio and Carlos, 2007) can solve these above-mentioned problems. Parallelization enables particles among each sub-population to be independent, thereby the population diversity is ensured. In the meanwhile, parallel particle swarm algorithms can improve the convergence accuracy. Parallel algorithms need to divide the swarm into several independent sub-populations each other. PAM is one of the popular clustering algorithms and more robust than *k*-means in the presence of noise and outliers (Jiawei and Micheline, 2005). In addition, uniform design can make population members uniformly scatter over the feasible solution space. Therefore, this study introduces PAM and uniform design to the parallel particle swarm algorithm. The simulation results show that the proposed algorithm has higher performance, higher rate of convergence than traditional parallel PSO.

Corresponding Author: Jie Zhang, School of Computer Science and Technology, Xidian University, Xi'an 710071, P.R. China

PARTICLE SWARM OPTIMIZATION ALGORITHMS

Standard particle swarm optimization algorithm: Assuming to optimize a continuous function containing d variables, search space should be d-dimensional. The ith particle's position and velocity respectively, is expressed as: $X_i = [x_{i,1}, x_{i,2},\ldots, x_{i,d}]$ and $V_i = [v_{i,1}, v_{i,2},\ldots, v_{i,d}]$. The particle's optimal position of the kth iteration is (Pbest) $P_i = [p_{i,1}, p_{i,2},\ldots, p_{i,d}]$, optimal position of swarm is (Gbest) $P_g = [p_{g,1}, p_{g,2},\ldots, p_{g,d}]$. Particles of the population update velocity and position in accordance with the following formulas:

$$v_{ij}(k+1)=w\cdot v_{ij}(k)+c_1\cdot r_1\cdot(p_{ij}(k)-x_{ij}(k))+c_2\cdot r_2\cdot(p_{gj}(k)-x_{ij}(k)) \quad (1)$$

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

where, the parameter w is the inertia weight coefficient, which indicates the particle's ability to maintain the last speed. The acceleration coefficient $c1$, $c2$ represent the degree of tracking individual optimal and global optimal respectively. The parameters $r1$ and $r2$ indicate uniformly distributed random numbers between 0 and 1.

The update equation of velocity consists of three components, including the previous velocity component, a cognitive component and a social component. They are mainly controlled by three parameters: the inertia weight and two acceleration coefficients. Considering different parameter values have different influence on the flying behavior of particle directly, so we call them as the behavior parameter set here.

From the theoretical analysis of the trajectory of a PSO particle (Clerc and Kennedy, 2002), the trajectory of a particle X_i converges to a weighted mean of P_i and P_g. Whenever the particle converges, it will "fly" to the individual best position and the global best position. According to the update equation, the individual best position of the particle will gradually move closer to the global best position. Therefore, all the particles will converge onto the global best particle's position. In fact, the particles usually converge on a local optimum.

Some improvements of PSO algorithm: For a practical swarm optimization problem, we often need a better global search to help the algorithm converge to an area quickly and then we need a stronger local search to get high precision solution. The inertia weight parameter has greatly influence on both global search and local search; so many attempts have been tried by using various inertia weight strategies to make its performance better (Shi and Eberhart, 1999; Chatterjee and Siarry, 2004; Chen *et al*., 2006). In order to balance between the global and local exploration abilities and obtain a quick search, it is necessary to dynamically adjust the inertia weight. Shi and Eberhart (1999) firstly have proposed a linearly decreasing inertia weight instead of a fixed inertia weight. By linearly decreasing the inertia weight from a relatively large value to a small value through the run, PSO tends to have better global searching ability at beginning of the run while having finer local search near the end of run. The inertia weight is adjusted with the following formula:

$$w^{(k)} = w_{max} - k(w_{max} - w_{min})/k_{max}$$

where,

$w_{\max}$	= The maximal value
$w_{\min}$	= The minimal value of w
k	= The iteration number, with
$k_{\max}$	= as its maximal value in a run

In this strategy, the particles could move fast in the beginning of the search, so that a sufficient optimal region can be quickly detected. The inertia weight changes with the iteration number k, so that the particles will slow down to search the local region. The strategies of nonlinear inertia weight were reported. An improved strategy was proposed (Chatterjee and Siarry, 2004):

$$w^{(k)} = (w_{max} - w_{min}).[(k_{max} - k)/k_{max}]^n + w_{min}$$

Another algorithm was proposed as follows (Chen *et al*., 2006):

$$w^{(k)} = w_{\min} + (w_{\max} - w_{\min}) \cdot e^{-k/(\frac{k_{\max}}{10})}$$

and

$$w^{(k)} = w_{\min} + (w_{\max} - w_{\min}) \cdot e^{-[k/(\frac{k_{\max}}{4})]^2}$$

All parameters in the aforementioned equations share the same meanings with the parameters in the linear decreasing strategy.

A PSO algorithm with self-adaptive and random inertia weight has been proposed (Zhang *et al*., 2005). The algorithm selects the inertia weight randomly according to the variance of the population's fitness σ as the following formula:

$$w=\begin{cases} 0.5+\dfrac{rand()}{2.0}, \sigma \geq 1.0 \\ 0.4+\dfrac{rand()}{2.0}, \sigma < 1.0 \end{cases}$$

From the formula we can see that w is a random number between (0.4, 1) and doesn't decrease linearly.

It has been proposed that the inertia weights are replaced by chaotic map parameter. The chaotic decreasing inertia weight and the chaotic stochastic inertia weight are proposed (Yong *et al*., 2007a, b). A method, of which random numbers r_1 and r_2 is replaced by chaotic map, are proposed (Jiang *et al*., 2005). The PSO algorithms which combined chaos search strategy are proposed (Yaoyao *et al*., 2008; Xue-Yao *et al*., 2008; Zhenglei *et al*., 2007). The same features of these strategies are as follows. First of all, the chaotic interval variables are mapped to the interval of the optimized variable by chaotic mapping. And then, chaotic local search is carried out within all particles, or some of the superior particles, or in the vicinity of the current global optimal solution. The particle swarm optimization algorithm based on dynamic chaotic perturbations is proposed in our previous study (Jie *et al*., 2009). Dynamic chaotic perturbations can get rid of blindly searching and decrease iteration times, compared with the multiple local chaotic searching.

Multiple populations or swarms: Many researchers have considered multi-populations as a means of enhancing the diversity of EAs. A self organizing scout was proposed (Branke *et al*., 2000). It has been shown to give excellent results on many peaks benchmark. Zeng *et al.* (2005) proposed an orthogonal design based evolutionary algorithm, called ODEA, where its population consists of “niches” and an orthogonal design method is employed. ODEA borrows some ideas from the SOS algorithm, however, the experimental results show that the performance of ODEA is better than the SOS algorithm. A speciation based PSO (SPSO) was developed (Parrott and Li, 2004), it dynamically adjusts the number and size of swarms by constructing an ordered list of particles, ranked according to their fitness, with spatially close particles joining a particular species.

THE PAM ALGORITHM

There are many clustering methods available in data mining. Typical clustering analysis methods are clustering based on partition, hierarchical clustering, clustering based on density, clustering based on grid and clustering based on model.

The most frequently used methods of clustering based on partition are *k*-means and *k*-medoids. In contrast to the k-means algorithm, *k*-medoids chooses data points as centers, which make *k*-medoids method more robust than k-means in the presence of noise and outliers. The reason is that a medoid is less influenced by outliers or other extreme values than a mean. Partitioning around medoids, abbreviated PAM is the first and the most frequently used *k*-medoids algorithms, as shown in Fig. 1.

```
Algorithm: $k$-medoids. PAM, a $k$-medoids algorithm for partitioning based on medoid or central objects.
Input:
$k$: the number of clusters,
$D$: a data set containing $n$ objects.
Output: A set of $k$ clusters.
Method:
(1) Arbitrarily choose $k$ objects in $D$ as the initial representative objects or seeds;
(2) Repeat
(3) Assign each remaining object to the cluster with the nearest representative object;
(4) Randomly select a nonrepresentative object, donated as orandom;
(5) Compute the total cost, $S$, of swapping representative object, oj, with orandom;
(6) if $S < 0$ then swap oj with orandom to form the new set of $k$ representative objects;
(7) Until no change.
```

Fig. 1: PAM, k-medoids partitioning algorithm

PAM constructs k partitions (clusters) of the given dataset, where each partition represents a cluster. Each cluster may be represented by a centroid or a cluster representative which is some sort of summary description of all the objects contained in a cluster. It needs to determine k partitions for n objects. The process of PAM is by and large as follows. Firstly, randomly select k representative objects and cluster other objects to the same group as the representative object according to the minimum value of the distances between the representative object and other objects. Then try to replace these non -representative objects with other non-representative objects, in order to minimize squared error. All the possible pairs of objects are analyzed, where one object in each pair is considered a representative object and the other is not. The total cost of the clustering is calculated for each such combination. An object, o_j, is replaced with such a object minimizing squared error. The set of best objects in each cluster after iteration forms the representative objects for the next iteration. The final set of representative objects are the respective medoids of the clusters.

UNIFORM DESIGN

Uniform array and uniform design: In this section, we briefly describe an experimental design method called uniform design. The main objective of uniform design is to sample a small set of points from a given set of points, such that the sampled points are uniformly scattered. The more details can be referred to the study (Leung and Wang, 2000).

Let there be n factors and q levels per factor. When n and q are given, the uniform design selects q combinations out of q^n possible combinations, such that these combinations are scattered uniformly over the space of all possible combinations. The selected q combinations are expressed in terms of a *uniform array* U (n, q) = $[U_{i,j}]_{q \times n}$, where $U_{i,j}$ is the level of the jth factor in the ith combination and can be calculated by the following formula.

$$U^{i,j} = (i\sigma^{j-1} \bmod q) + 1 \tag{3}$$

where, σ is a parameter given in Table 1.

Table 1: Values of the parameter σ for different number of factors and different number of levels per factor

Number of levels per factors	Number of factors	σ
5	2~4	2
7	2~6	3
11	2~10	7
13	2	5
	3	4
	4~12	6
17	2~16	10
19	2~3	8
	4~18	14
23	2,13~14, 20~22	7
	8~12	15
	3~7, 15~19	17
29	2	12
	3	9
	4~7	16
	8~12,16~24	8
	13~15	14
	25~28	18
31	2,5~12,20~30	12
	3~4,13~19	22

Improved Initial Population Based on Uniform Design Leung and Wang (2000) had designed a algorithm for dividing the solution space and a algorithm for generating the initial population. However, algorithm 2 only considers the dividing of the solution space, not does the dividing of the N-dimension space. This will bring about some serious problems. If we assume that in step 2, $Q_0 = 5$ and $N = 10$, then U(N, Q_0) is impossible to be generated because Q_0 must be larger than N. Namely, algorithm 2 is only fit for the low dimension problem, not fit for the high dimension problem. In order to overcome the shortcomings, we introduce the dividing of the N-dimension space. Therefore, algorithm 2 is modified as follows.

Step 1: Judge whether Q_0 is valid or not, as it must be found in the 1st column of Table 1. If no, stops and shows error messages, otherwise continues.

Step 2: Execute Algorithm 1 to divide [l, u] into S subspaces [l (1), u (1)], [l (2), u (2)],… [l (S), u (S)].

Step 3: Judge whether Q_0 is more than N, if yes, turns to step 4, otherwise turns to step 5.

Step 4: Quantize each subspace, and then apply the uniform array U (N, Q_0) to sample Q_0 points.

Step 5: We divide N-dimension space into $\lfloor N/N_0 \rfloor$ parts, where N_0 is an integer more than 1 and less than Q_0 and are generally taken as Q_0-1. Among $\lfloor N/N_0 \rfloor$ parts, the 1st part corresponds to the dimension from 1 to N_0 and the 2nd part corresponds to the dimension from N_0+1 to 2*N_0 and so forth. If the remainder R of N/N_0 isn't equal to 0, then a plus part corresponds to the dimension from $\lfloor N/N_0 \rfloor * N_0+1$ to N, whose length is surely less than N_0. Repeat to execute step 6 for each part.

Step 6: Quantize each subspace, and then apply the uniform array U (N_0, Q_0) to sample Q_0 points. In the plus part, U (N_0, Q_0) is replaced with U(R, Q_0), where the remainder R is equal to $N-\lfloor N/N_0 \rfloor * N_0$.

Step 7: Based on each fitness function, evaluate the quality of each of the SQ_0 points generated in step 2, and then select the best $\lfloor G/D_0 \rfloor$or $\lceil G/D_0 \rceil$ points. Overall, a total of G points are selected to form the initial population. For the single objective problem, D_0 is taken as 1, therefore, directly select the best G points.

IMPROVED PARALLEL PSO BASED ON PAM AND UNIFORM DESIGN

Thoughts of algorithm: Most of the parallel particle swarm algorithms all adopted IPPSO (Is-land-based Parallel PSO) model. The whole particle swarm is divided into several sub-populations, within which the global PSO search is carried out. We calculate and evaluate the fitness of each particle and select the best as the optimal particle within the sub-population. Evolution of each sub-population is carried out by single child process. Each child process will adopt concentrated migration strategy and periodically distributed the best particle in its island to the main process. The main process will broadcast the global best particle to each child processes and impel them to do the global optimum evolution.

In this study, we will design a parallel PSO algorithms based on PAM and uniform design as follows. Each child process only communicates with the main process and the best particles within each sub-population are sent to the main process by corresponding child process. The main process compares all the received best particles within each sub-population and generates the best particle and the current optimal value within the whole swarm. The child process can also communicate with each other when communication is needed, rather than the main process broadcasts the best particle to every child process. This not only prevents premature convergence, but also enables each child process to initiatively communication with the main process and to reduce many blind communications.

According to Darwin's theory of the survival of the fittest, some worse individuals should be eliminated from the population as the population is being evolved and improved. Uniform design is used to fulfill the task.

We can first generate the population based on PAM and uniform design within the subinterval defined by the individual best. Then replace those worse individuals in the old population with those better ones in the new population. As the subinterval is a fraction of $|S|$ parts of the whole interval, if it is again divided $|S|$ parts, each subinterval will be a fraction of $|S|^2$ parts of the whole interval and so forth. Therefore, the search space will change smaller and smaller. This is very fit with the principles of from rough search to finely search. In the meanwhile, as the population is being evolved and improved, the population members are getting closer to each other, so that the search space defined by the individual best is becoming smaller. Since the number of the sample points is fixed, the quantized points are getting closer and hence we can get more and more precise results.

IPPSO often adopts the strategy that communication is carried on after a fixed iterating number or cycle. This strategy is simple and easy to do. However, the current optimal individual did not change or change very little after a fixed iterating number in some occasion, therefore this will cause ineffective communication and waste communication time. The study improves this situation as follows. Only in certain conditions or in necessary occasion will communications be carried on. Namely, after one cycle, only when the best value in sub-population is better than that in the entire swarm will communications between the child process and the main process be carried out.

Parallel particle swarm algorithms first need to divide the swarms into several sub-populations. IPPSO algorithms classify particles according to the index value of them. This causes the number of particles in each sub-population is equal and the position and velocity of every particle is completely different. However, in the real-world environments, the size of each swarm is seldom the same. At the meanwhile, the individuals with the same or similar feature may be easily to stay together, which may be so-called “Birds of a feather flock together”. Therefore, when dividing the particles, the particles with similar features should be clustered together to form a sub-population and the particles with different features should be divided into different swarm. This is to enable the particles within the same sub-population to be more easily learned from each other. In clustering algorithm, PAM algorithms possess the features of high efficiency, high expandability and being easily to be implemented. Therefore, PAM is chosen as clustering algorithm to form sub-population.

Algorithm steps:

Step 1: According to the population N, determine the population size Q_0 in each subinterval and the number S of subintervals, such that Q_0*S is more than or equal to N, Q_0 is a prime and must exist in Table 1. Execute *Algorithm* 3 to generate an initial population *pop*.

Step 2: Evaluate each particle in *pop* and then acquire the individual best *P best* and the global best value *G best*. Then, assign *G best* to the main process.

Step 3: Cluster *pop* to form *K* sub-population using *PAM*. Each sub-population is responsible for one child process.

Step 4: Update each particle's position and velocity according to the formulas (1) and (2) separately within each sub-population.

Step 5: Within each sub-population, evaluate each particle to acquire the individual best *P best* and the local best *L best* of in its sub-population.

Step 6: If a given period (number of iterations), is called as the migration period, *T*, has reached and then compare *L best* and *G best*. If *L best* is better than *G best*, then communications are carried out between the main process and the child process, otherwise the iterating time without communications is saved to the variable *Tno*. Among, the steps of communication are as follows:
Firstly, the worst particle of the child process is replaced with *G best* of the main process and then the child process transfer *L best* to the main process to replace *G best*.

Step 7: If *Tno* of the child process reaches a predefined number (such as 2), then force to carry out one-way communication as follows. Firstly, replaces the worst particle with *L best* of the child process and then receives the global best *G best* of the main process to replace *L best* of the child process.

Step 8: If the migration period *T* reaches the times of certain predefined number (such as 5), then merge *K* sub-population into one swarm. The population and the individual best after merging are marked as *Lpop* and *LP best*, respectively.

Step 9: Find the minimum value and the maximum value of each dimension of *LP best* to generate a new search space [*LTmp*, *UTmp*]. Execute *Algorithm* 3 to generate a new population *popTmp* in the new search space. According to *Algorithm* 3, *popTmp* is sorted in ascending order, therefore the best particle locates in the first position and the second best one locates in the second position and so forth. Sort *Lpop* and *LP best* in descending order in term of the fitness of *LP best*, therefore the worst particle locates in the first position and the second worst one locates in the second position and so forth.

Step 10: Loop the following step: If the fitness of the ith particle in *popTmp* is less than or equal to that of the ith one in *LPbest*, then the ith particle in *LPbest* and *Lpop* will be replaced the ith particle in *popTmp*, where $i = 1,2,\ldots, M$, which indicates the population size. Otherwise break loop. The purpose of the step is to replace those worse individuals in *Lpop* and *L Pbest* with those better ones in *popTmp*. There exist two extreme cases. One of them is that all particles in *Lpop* and *LPbest* will be replaced and another is that none will be replaced. Compare the first particle in *popTmp* with *G best*, if the former is superior to the latter, then *G best* will be updated by the corresponding data of the first particle in *popTmp*.

Step 11: Cluster the swarm again to form K new sub-population by *PAM* and update the data information such as population size, the individual best and the local best of each new sub-population.

Step 12: Judge whether each sub-population is empty or not, if yes, divide half of the largest sub-population to the empty sub-population and update the corresponding information such as population size, the individual better and the local better of two sub-populations.

Step 13: If the stop criterion is satisfied, output the optimal solution *G best* and its optimal value, otherwise turns to step 4 and continues.

ALGORITHM SIMULATIONS

Four parallel PSO algorithms, IPPSO (Parallel PSO based on Island), PAMPPSO (Parallel PSO base on PAM), UPPSO (Parallel PSO base on Uniform design) and the proposed UPAMPPSO (Parallel PSO base on PAM and Uniform design), are respectively adopted to optimize several well-known Benchmarks problems and their performance are compared. Among the above algorithms, PAMPPSO and UPPSO are the algorithms of respectively getting rid of the uniform design and PAM parts of UPAMPPSO algorithm. Four algorithms are respectively carried independently out 30 runs to calculate the mean value and standard deviation of the optimal value.

Test problems: Several well-known Benchmarks problems are shown in Table 2, where f_1, f_2, f_3, f_4 and f_5 respectively indicates the function for Sphere, Rosenbrock, Griewank, Rastrigrin and Schwefel. Their theoretical optimal values are all 0 and dimensions are all assumed as 30.

Table 2: Test function

Function Expression	Search scopes
$f_1 = \sum_{i=1}^{n} x_i^2$	[-100,100]
$f_2 = \sum_{i=1}^{n-1} \left[100\,(x_{i+1} - x_i^2)^2 + (x_i - 1)^2\right]$	[-30,30]
$f_3 = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos(\frac{x_i}{\sqrt{i}}) + 1$	[-600,600]
$f_4 = \sum_{i=1}^{n} [x_i^2 - 10\cos(2\pi x_i) + 10]$	[-100,100]
$f_5 = \sum_{i=1}^{n} \lvert x_i \rvert + \prod_{i=1}^{n} \lvert x_i \rvert$	[-10,10]

Table 3: The mean value when k = 3

Fun	IPPSO	PAMPPSO	UPPSO	UPAMPPSO
f_1	1.11	1.39e-5	2.03e-6	4.51e-8
f_2	5.65	5.24	3.94	3.56
f_3	1.82e-1	8.88e-3	2.47e-2	7.31e-3
f_4	4.56	4.48	4.59	4.19
f_5	2.64	1.88	8.44e-5	9.81e-6

Table 4: The standard deviation when K = 3

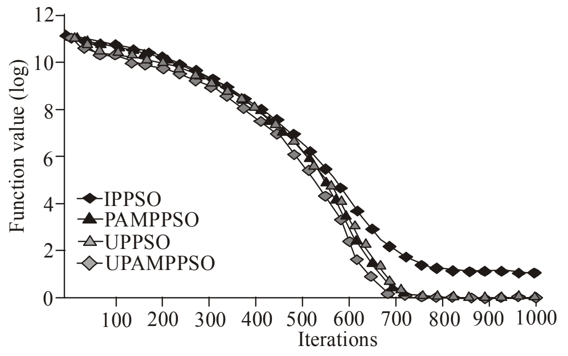
Fun	IPPSO	PAMPPSO	UPPSO	UPAMPPSO
f_1	2.88	3.07e-5	1.78e-6	3.27e-8
f_2	1.22	1.95	0.75	0.78
f_3	2.01e-1	6.87e-3	2.88e-2	8.31e-3
f_4	3.25 e-1	2.63e-1	3.21e-1	2.64e-1
f_5	3.87e-1	1.32	6.70 e-5	8.01e-6

Parameter values: The parameters of four parallel PSO algorithms are set as follows.

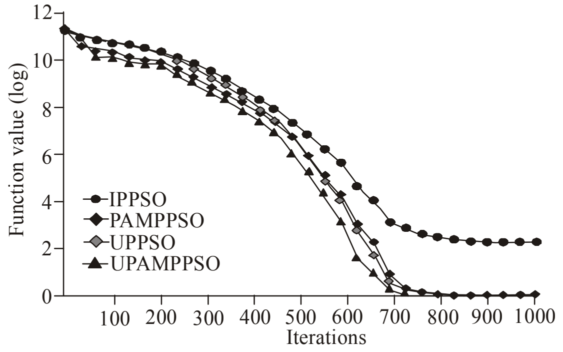
- **Parameters for PSO:** The linear descending inertia weight coefficient is within the closed interval [0.1, 1]. Population size is 40 and the acceleration coefficient c_1, c_2 are both taken as 2.
- **Parameters for parallel:** The migration period T is 10. In PAMPPSO and UPAMPPSO, when *Tno* of the child process is taken as 2, the one-way communication is forced to carry out. When the migration periods T are the times of 5, UPPSO, PAMPPSO and UPAMPPSO will carry out corresponding operating.
- **Parameters for uniform design:** The number of subintervals S is 4 and the number of the sample points or the population size of each subinterval Q_0 is 17.
- **Stopping condition:** the maximal generations are 1000.

RESULTS

When the numbers of sub-population K is equal to 3, the mean value and standard deviation of the optimal value are shown in Table 3 and 4 and the convergence processes of the mean value for f_1, f_2, f_3, f_4 and, f_5 are respectively shown in the left Fig. 2 to 6. When $K = 5$, the mean value and standard deviation of the optimal value are shown in Table 5 and 6 and the convergence processes of the mean value for f_1, f_2, f_3, f_4 and f_5 are respectively shown in the right Fig. 2 to 6.

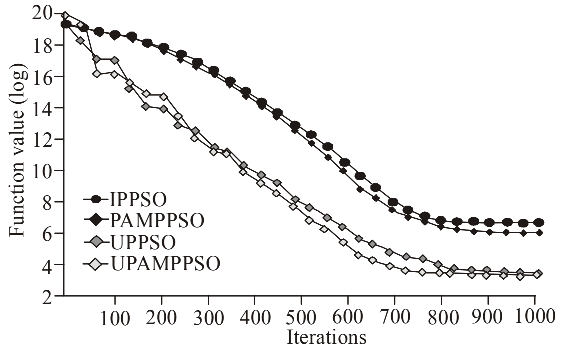


(a)

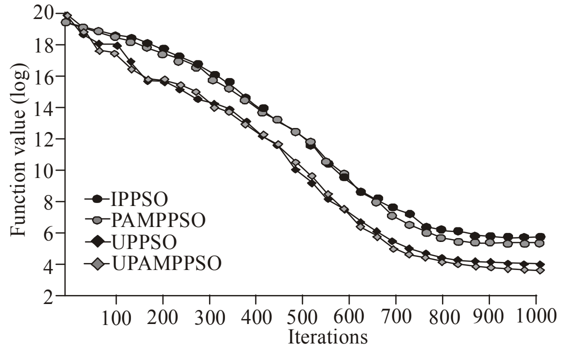


(b)

Fig. 2: The convergence process of the mean value for f_1

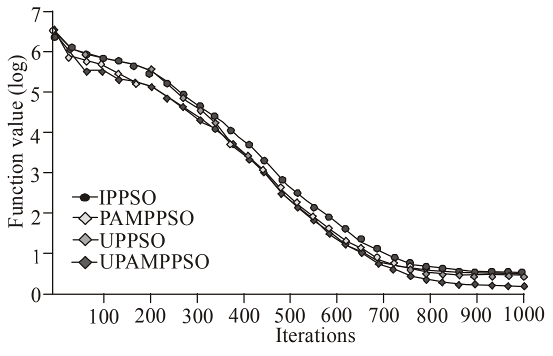


(a)

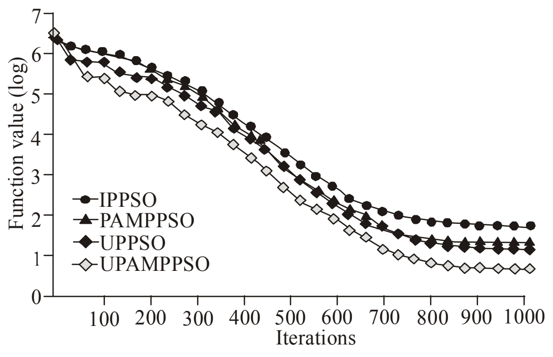


(b)

Fig. 3: The convergence process of the mean value for f_2

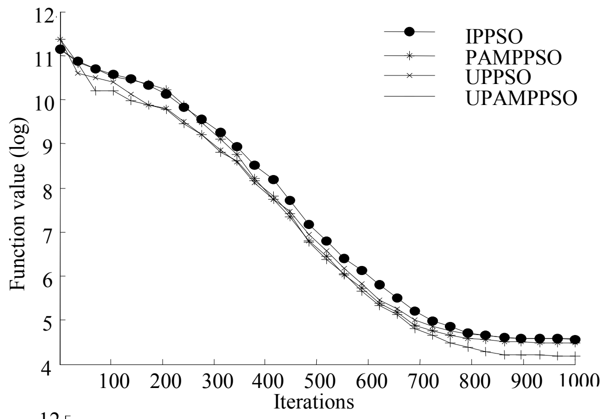


(a)

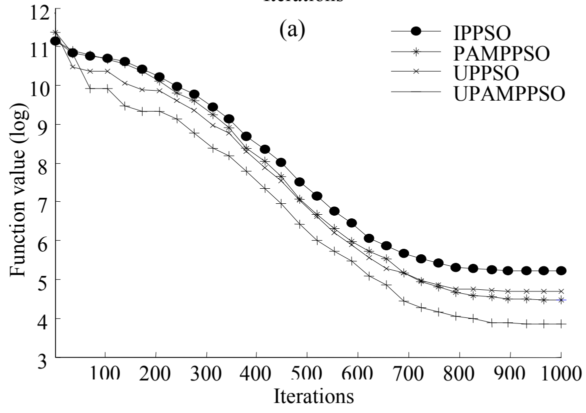


(b)

Fig. 4: The convergence process of the mean value for f_3

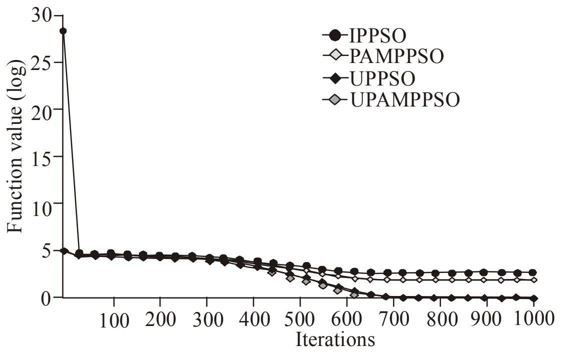


(a)

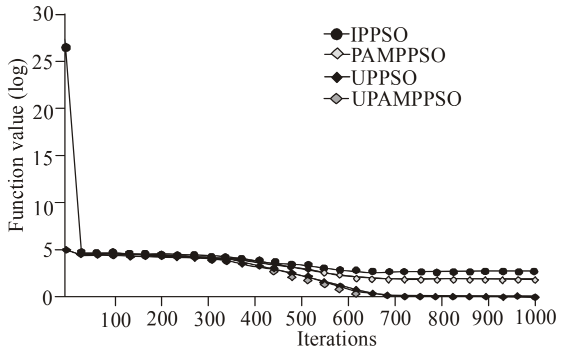


(b)

Fig. 5: The convergence process of the mean value for f_4



(a)



(b)

Fig. 6: The convergence process of the mean value for f_5

Table 5: The mean value when K = 5

Fun	IPPSO	PAMPPSO	UPPSO	UPAMPPSO
f_1	2.27	1.50e-4	2.14e-5	2.46e-7
f_2	6.63	5.95	3.43	3.25
f_3	7.02e-1	2.11e-2	1.27e-1	1.21e-2
f_4	5.21	4.47	4.69	3.85
f_5	3.01	1.84	1.41e-3	1845e-4

Table 6: The standard deviation when K = 5

Fun	IPPSO	PAMPPSO	UPPSO	UPAMPPSO
f_1	1.02	3.07e-4	1.82e-5	5.20e-7
f_2	7.99e-1	1.17	1.27e-1	1.05
f_3	1.83e-1	1.94e-2	7.58e-2	1.82e-2
f_4	2.39e-1	3.39e-1	1.63e-1	3.83e-1
f_5	3.11e-1	1.34	5.24e-4	1.66e-4

From Table 3 and 5, we can see that since the numbers of sub-population is 3 and 5 respectively, so the items in Table 5 converge more slowly than those in Table 3. This reason is that the more the number of sub-populations is, the less number of particles within each sub-population is and the fewer the opportunities for learning from each other within the sub-population are. In the migration cycle, each particle is impacted by the local optimal solution rather than the global optimal solution within its associated sub-population. Only when migration cycle is reached, information may be exchanged each other to obtain the global optimal solution. However, the particles among various sub-populations can not exchange information within the migration cycle. That is to say, the more the number of sub-populations is, the slower the convergence speed is and the easier to avoid premature convergence.

The proposed algorithm compared with IPPSO, there are much improvement on the $f_1 \sim f_5$, as shown in Table 3 to 4 and Fig. 2 to 6.

For f_1, the mean value of in Table 3 decreases by 8 orders of magnitude and that in Table 5 decreases by 7 orders of magnitude. The standard deviation is similar to the mean value. Figure 2a shows that the proposed algorithm inclines more quickly than IPPSO, especially in later periods of iterations. In Fig. 2a, the curve of the proposed algorithm is always below that of IPPSO and is almost parallel after 800 iterations. In Fig. 2b, there are a little difference after 20 iterations, apparent difference after 80 iterations and almost parallel after 700 iterations between the curves of the proposed algorithm and IPPSO.

For f_2, Table 3 and 4 show that the mean value and the standard deviation of the proposed algorithm are both better than in IPPSO, Table 5 is similar to Table 3. In Fig. 3a,b, there is distinct difference on the curve of the proposed algorithm compared with that of IPPSO after 50 iterations and almost parallel after 700 iterations.

There is a little similar in the mean value f_3 compared with f_1.The mean value of the proposed algorithm in Table 3 decreases by 2 orders of magnitude and that in Table 5 decreases by 1 order of magnitude. In Table 4 and 6, the variations of the standard deviation are similar to those of the mean value. In the Fig. 4a, there is distinct difference on the curve of the proposed algorithm compared with that of IPPSO after 100 iterations and almost parallel after 850 iterations. In Fig. 4b, there are apparent difference after 50 iterations and almost parallel after 800 iterations.

The results of f_4 are similar to that of f_2 in Table 3 and 5. The mean value and the standard deviation of the proposed algorithm are both significantly smaller than those of IPPSO. In Fig. 5a, there is distinct difference on the curve of the proposed algorithm compared with that of IPPSO after 100 iterations and almost parallel after 850 iterations. In Fig. 5b, there are apparent difference after 50 iterations and almost parallel after 800 iterations.

For f_5, the mean value of the proposed algorithm in Table 3 decreases by 6 orders of magnitude and that in Table 5 decreases by 4 orders of magnitude. The standard deviation of the proposed algorithm in Table 4 decreases by 5 orders of magnitude and that in Table 4 decreases by 3 orders of magnitude.

In Fig. 6a, there is distinct difference on the curve of the proposed algorithm compared with that of IPPSO after 450 iterations and almost parallel after 700 iterations. In Fig. 6b, there are apparent difference after 500 iterations and almost parallel after 700 iterations.

Table 7: The statistical value in different dimension

Dim.	IPPSO	PAMPPSO	UPPSO	UPAMPPSO
10	1.5721e-13	2.2204e-16	2.2204e-16	2.2204e-16
20	9.8645e-6	2.8209e-7	7.2922e-8	5.9779e-9
30	2.6418	1.8796	8.4421e-5	9.8139e-6
40	3.4779	2.3869	2.0144e-3	3.3103e-4
50	4.1685	3.3845	1.2108e-2	4.5766e-3
60	4.4364	3.7610	3.8842e-1	7.7851e-2
70	4.7911	4.0950	2.8510e-1	2.1207e-1
80	4.9808	4.8673	1.0999	2.3619e-1
90	5.2878	4.7461	1.6481	9.0177e-1
100	5.3292	5.2586	2.7576	2.5298

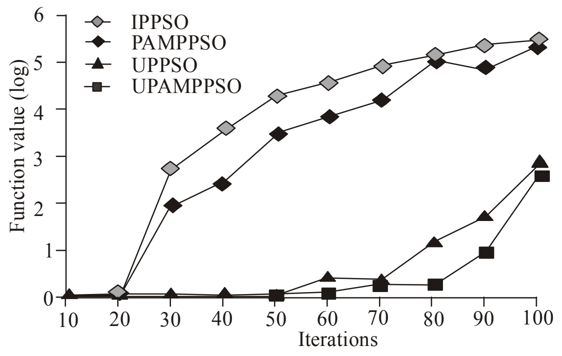


Fig. 7: The mean value versus the dimension

From the abovementioned simulations results, a conclusion can be drawn that the proposed algorithm is much more stable than IPPSO and its rate of convergence is much faster than IPPSO.

Influence of dimension: In order to find out the influence of dimension, we carry out an experiment as follows. The dimension respectively is 10, 20, 30, 40, 50, 60, 70, 80, 90, 100 and other conditions keep unchanged. The test function f_5, namely Schwefel, is iterated 1000 time and independently carried out 30 runs to get its mean value. The final outputs are shown in Table 7. We assume the dimension as x axis, the mean value as y axis and the final curve figure is shown in Fig. 7.

Table 7 and Fig. 7 both show that the mean values of several algorithms both increase with increasing in dimension. However, the curve of UPAMPPSO in Fig.7 lies always in below that of the others and the mean value of UPAMPPSO is always the smallest in Table 7, which just indicates the proposed algorithm is superior to the others. Furthermore, this also indicates that the dimension has no influence on the performance of the proposed algorithm.

Influence of Q_0 and S in uniform design: In order to find out the influence of Q_0 and S in the uniform design, we carry out an experiment as follows. The test function f_4 with dimension of 30 and f_5 with dimension of 50 is respectively iterated 1000 time using UPPSO and independently carried out 30 runs to get its mean value. The final outputs are shown in Table 8. We assume Q_0 as x axis, the mean value as y axis and the final curve figure is shown in Fig. 8.

Table 8: The statistical value

Q_0	f_4, $S = 8$	f_4, $S = 16$	f_4, $S = 32$	f_5, $S = 8$	f_5, $S = 16$	f_5, $S = 32$
5	4.0	4.5	4.2	8.6e-2	3.3e-2	1.1e-2
7	4.9	4.8	3.8	4.9e-2	6.7e-2	1.2e-2
11	4.3	4.8	4.5	3.0e-2	3.2e-1	1.1e-2
13	4.5	4.3	4.3	6.0e-3	8.8e-3	1.9e-3
17	3.9	3.8	4.8	2.0e-2	8.3e-3	6. 6e-3
19	10.3	10.3	10.3	4.1	4.1	4.1
23	9.9	9.9	9.9	3.7	3.7	3.7
29	11.7	11.7	11.7	5.2	5.1	5.1
31	11.7	11.7	11.7	5.3	5.1	5.1

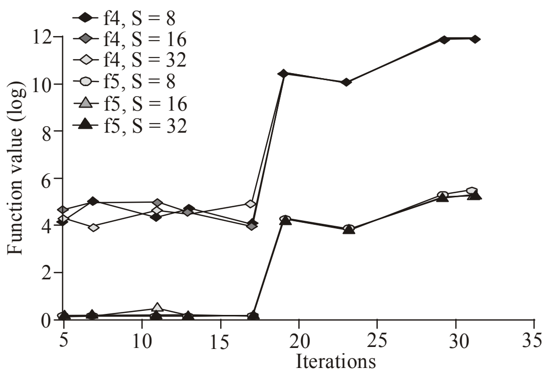


Fig. 8: The mean value of f_4 amd f_5 with different Q_0 and S

From Table 8 and Fig. 8, we can see that its mean values when Q_0 = 5, 7, 11, 13 and 17 are greatly superior to those when Q_0 = 19, 23, 29 and 31. This can guide how to choose the parameters Q_0. At meanwhile, for the same function, when Q_0 *is* kept unchanged, the function values of different S are close to each other. This indicates that the influence of S is far fewer compared with Q_0, therefore, the influence of S may be ignored in the uniform design. A good Q_0 greatly influence the performance of the uniform design, therefore, it is urgent how to choose a good Q_0. From Table 8 and Fig. 8, we can also see that the mean values when Q_0 = 5, 7, 11, 13 and 17 have little difference; therefore, any of them can be taken a good Q_0.

CONCLUSION

From Thoughts of the algorithm and simulation, we can see the effects of the proposed algorithm are better than those of IPPSO and closer to the theoretical optimal value. The proposed algorithm increases re-clustering when migration cycle has reached a certain value, but the improvement of algorithm performance is very obvious. This shows that when the position and speed have varied re-clustering is very necessary for a certain time. Meanwhile, uniform design has great influence on the improvement of algorithm performance. The simulation results just confirm this conclusion.

From another point of view, a crude search is needed in the iterating early period and a fine search is needed in the iterating later period. Similar particles are clustered together and easier to learn from each other by PAM. Namely, PAM carry out the function of a fine search. Meanwhile, the uniform design can carry out the function of the crude search. The combination of the two methods just serves as the combination of the crude search and the fine search, therefore, the proposed algorithm certainly outperform IPPSO algorithm.

ACKNOWLEDGMENT

This study is supported by the National Natural Science Foundation of China (No.61272119, No.61203372). Ph.D. Programs Foundation of Ministry of Education of China (No.20090203110005). The Fundamental Research Funds for the Central Universities (No. K50510030014).

REFERENCES

Antonio, L.J. and A. Carlos, 2007. Coello Coello. MRMOGA: A new parallel multi-objective evolutionary algorithm based on the use of multiple resolutions. Concurr. Comput. Pract. Exp., 19: 397-441.

Branke, J., T. Kaußler, C. Schmidth and H. Schmeck, 2000. A multi-population approach to dynamic optimization problems. Proceeding 4th International Conference on Adaptive Computing in Design and Manufacturing, pp: 299-308.

Chatterjee, A. and P. Siarry, 2004. Nonlinear Inertia weight variation for dynamic adaptation in particle swarm optimization. Comp. Oper. Res., 33: 859-871.

Chen, G., X. Huang, J. Jia and Z. Min, 2006. Natural exponential lnertia weight strategy in particle swarm optimization. Proceedings of the 6th World Congress on Intelligent Control and Automation, pp: 3672-3675.

Clerc, M. and J. Kennedy, 2002. The particle swarm: Explosion,stability and convergence in a multi-dimensional complex space. IEEE Trans. Evol. Comput., 6: 58-73.

Dudy, L., O. Yew-Soon, J. Yaochu, S. Bernhard and L. Bu-Sung, 2007. Efficient hierarchical parallel genetic algorithms using grid computing. Future Gen. Comp. Syst., 23: 658-670.

Eberhart, R. and J. Kennedy, 1995. A new Optimizer using particle swarm theory. Proceedings of 6th International Symposium Micro Machine and Human Science, pp: 39-43.

Jiang, C.W., B. Etorre and J. Jiang, 2005. A self-adaptive chaotic particle swarm algorithm for short term hydroelectric system scheduling in deregulated environment. Energy Conv. Manag., 46: 2689-2696.

Jiawei, H. and K. Micheline, 2005. Data Mining: Concepts and Techniques. Elsevier Inc., Oxford, UK, pp: 402-403.

Jie, Z., Y. Yajuan and Z. Quanju, 2009. The particle swarm optimization algorithm based on dynamic chaotic perturbations and its application to K-means. International Conference on Computational Intelligence and Security (CIS), Beijing, China, pp: 282-286.

Kennedy, J. and R. Eberhart, 1995. Particle swarm optimization. Proceedings of IEEE International Conference on Neutral Networks, pp: 1942-1948.

Leung, Y.W. and Y. Wang, 2000. Multiobjective programming using uniform design and genetic algorithm. IEEE Trans. Syst. Man Cy. C, 30: 293-304.

Parrott, D. and X. Li, 2004. A particle swarm model for tracking multiple peaks in a dynamic environment using speciation. Proceeding of the IEEE Congress on Evolutionary Computation, pp: 98-103.

Shi, Y. and R. Eberhart, 1999. Empirical study of particle swarm optimization. Proceedings of the IEEE Congress on Evolution Computation, Piscataway, NJ, pp: 1945-1950.

Xue-Yao, G., S. Li-Quan, Z. Chun-Xiang and Y. Shou-Ang, 2008. A modified particle swarm optimization algorithm based on improved chaos search strategy. International Symposium on Computational Intelligence and Design, pp; 331-335.

Yaoyao, H., Z. Jianzhong, L. Chaoshun, J. Yang and Q. Li, 2008. A precise chaotic particle swarm optimization algorithm based on improved tent map. 4th International Confernce on Natural Computation, pp: 569 -573.

Yong, F., T. Gui-Fa, W. Ai-Xin and Y. Yong-Mei, 2007a. Inertia weight in particle swarm optimization. 2nd International Conference on Innovative Computing, Information and Control, pp: 475-478

Yong, F., T. Gui-Fa, Y. Yong-Mei and W. Ai-Xin, 2007b. Comparing with chaotic inertia weights in particle swarm optimization. Proceedings of the 6h International Conference on Machine Learning and Cybernetics, pp: 329-333.

Zeng, S., H. De Garis, J. He and L. Kang, 2005. A novel evolutionary algorithm based on an orthogonal design for dynamic optimization problems. Proceeding of the 2005 IEEE Congress on Evol, Comput., 2: 1188-1195.

Zhang, X., S. Wen and H. Li, 2005. A novel particle swarm optimization algorithm with self-adaptive inertia weight. Proceedings of the 24th Chinese Control Conference, Guangzhou, P.R. China, pp: 1373-1376.

Zhenglei, Y., Y. Liliang, W. Yaohua, L. Liao and G. Li, 2007. Chaotic particle swarm optimization algorithm for traveling salesman problem. Proceedings of the IEEE International Conferenc on Automation and Logistics, pp: 1121-1124.