Hu XM, Zhang J, Li Y. Orthogonal methods based ant colony search for solving continuous optimization problems. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 23(1): 2–18 Jan. 2008

Orthogonal Methods Based Ant Colony Search for Solving Continuous Optimization Problems

Xiao-Min Hu[1,2] (胡晓敏), Jun Zhang[1,2,*] (张 军), and Yun Li[3] (李 耘)

[1] *Department of Computer Science, Sun Yat-Sen University, Guangzhou 510275, China*

[2] *Key Laboratory of Digital Life (Sun Yat-Sen University), Ministry of Education, Guangzhou 510275, China*

[3] *Department of Electronics and Electrical Engineering, University of Glasgow, Glasgow G12 8LT, Scotland, U.K.*

E-mail: junzhang@ieee.org

Revised November 6, 2007.

Abstract Research into ant colony algorithms for solving continuous optimization problems forms one of the most significant and promising areas in swarm computation. Although traditional ant algorithms are designed for combinatorial optimization, they have shown great potential in solving a wide range of optimization problems, including continuous optimization. Aimed at solving continuous problems effectively, this paper develops a novel ant algorithm termed "continuous orthogonal ant colony" (COAC), whose pheromone deposit mechanisms would enable ants to search for solutions collaboratively and effectively. By using the orthogonal design method, ants in the feasible domain can explore their chosen regions rapidly and efficiently. By implementing an "adaptive regional radius" method, the proposed algorithm can reduce the probability of being trapped in local optima and therefore enhance the global search capability and accuracy. An elitist strategy is also employed to reserve the most valuable points. The performance of the COAC is compared with two other ant algorithms for continuous optimization — API and CACO by testing seventeen functions in the continuous domain. The results demonstrate that the proposed COAC algorithm outperforms the others.

Keywords ant algorithm, function optimization, orthogonal design

1 Introduction

Recently, swarm computation has received an increasing attention in computing science. A significant benchmark application is the modeling of real ants in foraging behavior. One of the most notable experiments about ants' behavior is the double bridge experiment[1,2]. By sensing the pheromone existing in the environment, the ants are able to find the shortest path from their nest to the food. Such behavior inspires a great deal of effort in developing ant algorithms for solutions of different problems.

Since the first ant algorithm, the ant system (AS)[3], was proposed by Dorigo in his Ph.D. dissertation in 1992, it has subsequently developed into a paradigm of the ant colony optimization (ACO) metaheuristic[3]. It employs some basic heuristics in order to escape from local optima and to find the global optimum in a multimodal space. The most successful applications of ant algorithms are combinatorial optimization and network problems, such as the traveling salesman problem (TSP)[4], the vehicle routing problem (VRP)[5,6], the job shop scheduling problem (JSP)[7], the water distribution system (WDS)[8], data mining[9] and network routing[10], etc. However, all these algorithms are designed for solving discrete optimization problems.

Recently, research has been carried out to extend ant algorithms for solving continuous optimization problems. The first method is the continuous ACO (CACO), proposed by Bilchev and Parmee in 1995[11]. It was the rudiment of CACO and later it was consummated in [12, 13]. The CACO combines both ant colony and genetic algorithm (GA)[14] techniques together. Monmarché *et al.*[15] proposed a method termed API in 2000, where the ant colony search mechanism consists of a set of parallel local searches on hunting sites with sensitivity to successful sites and the ants' nest is periodically moved[15].

Regular Paper

Supported by the National Natural Science Foundation of China under Grant No. 60573066, the Guangdong Natural Science Foundation Research under Grant No. 5003346, and the Scientific Research Foundation for the Returned Overseas Chinese Scholars, State Education Ministry, P.R. China.

*Corresponding Author

Other methods developed recently include the continuous interacting ant colony (CIAC) algorithm[16,17], the extended ACO to continuous domains (ACO_R)[18,19], the continuous ant colony system (CACS)[20], the binary ant system (BAS)[21], the direct application of ACO (DACO)[22], and those algorithms reported in [23, 24]. Hybrid ant algorithms combined with genetic algorithms[23], artificial immune systems[25] and particle swarms (PSACO)[26] have also been proposed. These methods are able to tackle continuous optimization problems to a certain degree.

In order to solve continuous optimization problems effectively, this paper develops a continuous orthogonal ant colony (COAC) algorithm by using the orthogonal design method[27−32]. The orthogonal design method, which was proposed more than fifty years ago, is an experimental design method and has since been widely applied in scientific research, manufacturing, agricultural experiments and quality management. It can be used for planning experiments and provide an efficient way to find a near-best sample for the multifactor experiments. As every variable in a problem can be regarded as a factor, the orthogonal design method can help solve optimization problems, e.g., [33–37]. Zhang and Leung[33] and Leung and Wang[34] applied the orthogonal design method in the crossover operation of a GA.

Different from the above approaches, however, the algorithm developed in this paper uses a novel operation termed "orthogonal exploration" to find the optimum. Here, in the problem domain, every ant in the colony lands in a region selected under the guidance of pheromone. The orthogonal design method is implemented for the ant to search for a better point in the region, whose size is determined by its radius. After all ants finish searching for better points, the desirability of each region is re-evaluated and an elitist strategy is implemented to retain the best. Better regions in the domain have a higher probability to survive and the pheromone in those regions is accordingly reinforced. Then the other regions are discarded and replaced by randomly generated regions.

The algorithm developed in this paper also implements one additional feature to make it more robust and faster. The radiuses of regions are adaptively shrunk or expanded according to the prevailing search status. If a better point is found, the radiuses of that region will be expanded, or else the radiuses will be shrunk. The expansion of the radiuses is to enhance diversity to the algorithm, while shrinking is to find a better niche to enhance accuracy.

The next section of this paper describes the transformation of discrete ant algorithms for continuous optimization problems. The third section contains a brief review of the orthogonal design method and discusses the principles on which the proposed algorithm will be based. In Section 4, the proposed COAC algorithm is presented in detail. Section 5 describes the additional feature of the proposed algorithm. Experimental results are reported in Section 6, where the performance of the proposed algorithm is compared with CACO and API by seventeen continuous optimization functions. Discussions on the convergent characteristics of the proposed algorithm are also made in Section 6. Finally, Section 7 concludes the paper.

2 ACO to the Continuous Domain

Traditional ant colony optimization (ACO) is a framework for discrete optimization problems. The agents in ACO work as teammates in an exploration team. The cooperation between the ants is based on pheromone communication. In view of the traveling salesman problem (TSP), which is a discrete optimization problem, the mission for the agents is to find the shortest way to traverse all the cities and return to the original city. The number of roads inter-connecting the cities is limited and fixed. However, optimization problems in the continuous domain are different. There are no longer "fixed" roads for agents to explore, but walking in any directions in the n-dimensional ($n > 1$) domain can lead to quite a different result. The difficulty in solving such a problem lies in how to find an efficient way to direct these agents to search in the n-dimensional domain.

2.1 Discretizing the Continuous Domain

Consider a group of m agents, whose mission is to find the lowest point in the given domain. The simplest way is to drop the agents in the domain randomly and record the best point. Then a new group of m agents are dropped randomly to the domain again. After several times, the best location ever been found is taken as the result. This is a purely a-posteriori random search procedure and no heuristics is implemented in this method. Using a discrete optimization algorithm to solve a continuous algorithm, the continuous domain needs to be discretized.

When an ant locates in a domain, it stands at a point of which the desirability is corresponding to the optimization degree. The desirability is usually evaluated by the objective function modeling from the original problem. The continuous domain can be discretized into several regions and the ants can ex-

plore the regions instead of the whole domain. In this paper, we assign each region i a property, radius $\boldsymbol{r}_i = (r_{i1}, r_{i2}, \ldots, r_{in})$ (where n being the dimensions of the domain), to control its range. The radius in this paper is used by the orthogonal exploration that will be detailed later. In a large and high-dimensional domain, the division of the domain is difficult and incomplete. For simplicity, this paper randomly generates initial regions in the domain. The number of regions in this paper is defined as μ. The regions can be moved according to the optimization process. An example of placing three ants in four regions is shown in Fig.1.

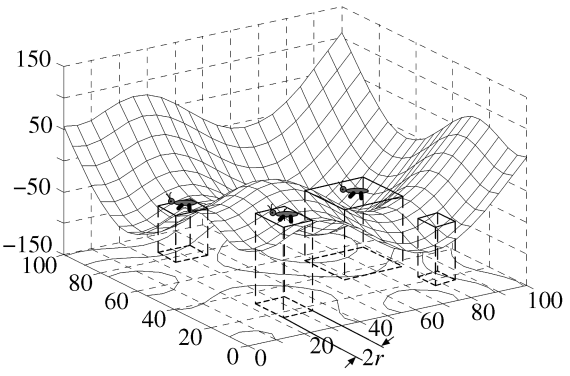


Fig.1. Example of placement of three ants in four regions in a two-dimensional domain.

2.2 Communication Between the Ants

The core in ACO is the use of pheromone. An ant can drop a certain amount of pheromone on the way. The more ants choose the way to go, the more pheromone accumulates on the road. If the road is good, chances are that more ants will select the good road and, after some time, the roads with the densest pheromone converge. In the continuous domain, the ideal situation is that the agents gather in the optimum place.

The artificial ants in this paper behave slightly differently from real ants. It is supposed that if an ant found out a good point, it would add pheromone to that region, or else nothing would be contributed to the region. In the continuous problem, each artificial ant is able to sense the amount of pheromone in all of the existed regions. The regions are randomly generated with radiuses or they are bequeathed from the past. The ants base on a rule, which defines an exact way and a probabilistic way, to select the regions to explore. The rule will be discussed in Section 4. Once in a region, the ant will further explore the area of the region to find a better point. The size of the region influences the accuracy that the ant can gain. In this paper, an orthogonal design method is used to guide the search to the regions fast and completely.

In the real world, the amount of pheromone will evaporate in the course of time. If ants do not explore the region, the pheromone in that region will be decreased. If the amount of pheromone in a region is too small, the region will be deserted. The orthogonal method to be developed in this paper will be based on the above background through applying ant colony heuristics so as to make the algorithm search efficiently in the continuous domain.

3 Orthogonal Design Method

The orthogonal design method is a way of planning experiments and it has been widely used in multifactor experiments. In an experiment, the parameters are termed factors, while the values of these parameters are termed levels. Consider an experiment with k factors and each factor with s levels. Then there will be s^k combinations to test. This is a full-scale experiment and the computation cost exponentially increases when k and s become larger. In order to reduce the number of experiments to make the problem tractable, the orthogonal design method is used here.

3.1 Orthogonal Design

The so-called orthogonal design is a method that makes complex designs of multifactor experiment feasible and compact. It samples a small, but representative set of combinations[29] by using a series of orthogonal arrays for arrangement of experiments[30]. Rao[27] used certain kind of orthogonal arrays in "fractional factorial" experiments[31] because of their desirable statistical properties in 1947. Since Bush[28] introduced the term "orthogonal array" in 1950, orthogonal arrays have become essential in statistics and are primarily used in designing experiments in most fields such as medicine, agriculture and manufacturing.

An orthogonal array, which varies from the size of the experiment, is used in the orthogonal design. We define $OA(N, k, s)$ as an orthogonal array with k factors and each factor with s levels, where N stands for the number of combinations to be tested. Take an orthogonal array $OA(9, 4, 3)$ as an example, which is shown in Table 1. Each column in the array stands for one factor, while the numbers in the column represent the levels of the factor. There are nine combinations of factors with different levels in the array, so that there are nine samples to be undertaken. The values in the level can be mapped with the actual values of

the parameters. Table 2 illustrates an example of experimental settings. The columns of A, B, C and D stand for factors. In column A, there are three levels and level 1 is mapped to 80°C, level 2 is mapped to 85°C, etc. The objective of the experiment is to find a best combination of such different factors to yield an optimal result. The use of the orthogonal array can schedule the experiment fast because there are only a few tests to be performed instead of a full-scale experiment. Though the experiment is partial, the result of the orthogonal experiment is still convincible for the principles of the orthogonal design.

Table 1. Orthogonal Array $OA(9, 4, 3)$

Combination	Factor 1	Factor 2	Factor 3	Factor 4
1	1	1	1	1
2	1	2	2	2
3	1	3	3	3
4	2	1	2	3
5	2	2	3	1
6	2	3	1	2
7	3	1	3	2
8	3	2	1	3
9	3	3	2	1

Table 2. Example of Experimental Settings for $OA(9, 4, 3)$

	Factors			
	A (°C)	B (kg)	C (type)	D (hour)
Level 1	80	35	Type1	1
Level 2	85	48	Type2	2
Level 3	90	55	Type3	3

3.2 Principles of Orthogonal Design

The number of combinations to be tested in the orthogonal design is much fewer than the full-scale experiment. Generally, the orthogonal design method is a partial experimental method to all the levels of factors, but it is a full-scale experiment to any two factors. The levels of the orthogonal array are made to be mostly "orthogonal" with each other. Take three factors to draw a cube, as shown in Fig.2. The three factors are denoted as A, B and C with subscripts indexing the levels. There are totally 27 combinations of three factors, which are illustrated as spots in Fig.2. Based on the first three columns of $OA(9, 4, 3)$, nine combinations are selected, which are illustrated as hollow spots in Fig.2. In the cube, there are three hollow spots on every face (including the inside faces) and one hollow spot on every edge (including the inside edges). These nine combinations can approximately reflect the solution space. Although the best combinations in these sampled experiments may not be the best one in the full-scale experiment, this method can reduce the number of tests and give a direction to the optimal combinations.

In fact, there are different types of orthogonal combinations with three factors. Any three columns without duplication of $OA(9, 4, 3)$ can form nine orthogonal combinations which are composed of different spots in Fig.2. Any orthogonal arrays with equal or more columns than the number of factors in the given multifactor problem can be used. This means that an orthogonal array without some columns is still an orthogonal array. However, an orthogonal array with more factors is always accompanied with more combinations that will consume exponentially longer time to complete the experiment.

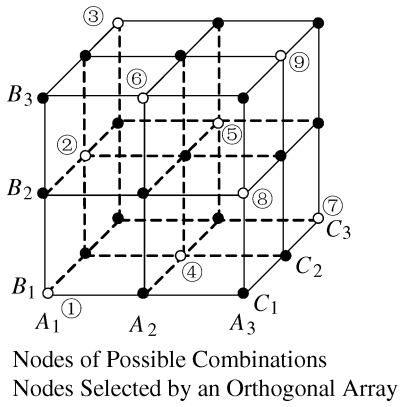


Fig.2. Distribution model of three factors with three levels.

An orthogonal array complies with the three elementary transformations. If any two factors of an orthogonal array are swapped, or any two levels of an orthogonal array are swapped, or any two levels of a factor are swapped, the resulting array is still an orthogonal array. In this paper, the columns of the orthogonal array used are randomly chosen in order to construct various kinds of orthogonal neighboring points in the proposed algorithm.

4 Orthogonal Ant Colony Optimization

The ants which are sent to find the optimal location in the given domain use pheromone and the orthogonal exploration to accomplish the mission. The domain is divided into multiple regions of various sizes. Every region has multi-properties as the searching radiuses, the coordinate of the center, the amount of pheromone, and the ranks by its desirability. The desirability is

evaluated by the objective function.

4.1 Orthogonal Exploration

In each iteration, m ants are dispatched. There are μ regions in the domain, which are randomly generated or inherited from the previous iteration. Based on the amount of pheromone in the regions, the ants choose which region to explore first. A user-defined probability q_0 is used to determine whether to choose the region with the highest pheromone or the region selected by the roulette wheel selection method. The rule an ant chooses region j is given by

$$j = \begin{cases} j|\max_{j\in S_R}\tau_j, & \text{if } q \leqslant q_0, \\ \text{RWS}, & \text{otherwise}, \end{cases} \tag{1}$$

where S_R is the set of regions, τ_j is the pheromone value in region j, q is a uniform random number in [0, 1]. RWS stands for the roulette wheel selection. Here the roulette wheel selection is based on the pheromone value of the regions. In RWS, the probability of selecting a region j is given by

$$p(j) = \frac{\tau_j}{\sum_{i\in S_R}\tau_i}, \quad \text{if} \quad j \in S_R. \tag{2}$$

After an ant chooses a region, it applies the orthogonal design method to select some points in the region to explore. Suppose the ant is now located in the center of the region $\boldsymbol{x}_j = (x_{j1}, x_{j2}, \ldots, x_{jn})$ with a radius $\boldsymbol{r}_j = (r_{j1}, r_{j2}, \ldots, r_{jn})$. Here n is the dimension of the domain to be explored and it is also the number of factors in multifactor experiments. The fact that every dimension in the problem can be regarded as a factor is the basis for implementing the orthogonal design method in solving the problem. The levels of the factors (variables) are computed using the radiuses of the region.

Given an orthogonal array $OA(N, k, s)$, where k must satisfy $k \geqslant n$, the orthogonal neighboring points $\boldsymbol{x}_j^{(i)}$ $(i = 1, 2, \ldots, N)$ of the current point $\boldsymbol{x}_j = (x_{j1}, x_{j2}, \ldots, x_{jn})$ is defined as follows:

$$\boldsymbol{x}_j^{(i)} = (x_{j1}^{(i)}, x_{j2}^{(i)}, \ldots, x_{jn}^{(i)}) \tag{3}$$

where

$$\begin{aligned} x_{jp}^{(i)} &\leftarrow x_{jp} + \frac{(OA_{ip} - 1)\cdot 2 - 1}{s - 1}\cdot r_{jp}\cdot rand_j, \\ &\text{if} \quad x_{jp}^{(i)} < l_p, \quad \text{then} \quad x_{jp}^{(i)} \leftarrow l_p, \\ &\text{if} \quad x_{jp}^{(i)} > u_p, \quad \text{then} \quad x_{jp}^{(i)} \leftarrow u_p, \\ &p = 1, 2, \ldots, n, \quad k \geqslant n, \end{aligned} \tag{4}$$

and OA_{ip} is the level of the i-th combination to the corresponding column in the array. As the array composed of any n columns without duplication in an orthogonal array $(k \geqslant n)$ is still an orthogonal array, the columns used in the orthogonal exploration are selected randomly and form a new array with n columns.

By using the new orthogonal array, N orthogonal neighboring points are generated. According to the properties of an orthogonal array, the N selected points can approximately represent the characteristic of the surrounding region. Examples of the orthogonal points an ant selects in a two-dimensional region j with a radius $\boldsymbol{r}_j = (r_{j1}, r_{j2})$ using $OA(9, 4, 3)$ and $OA(16, 5, 4)$ are illustrated in Fig.3. The center hollow spot stands for the ant, while the solid spots stand for the orthogonal neighboring points. The value $rand_j$ in (4) is set to 1 in the figure for better explanation. Because it is a full-scale experiment to any two combinations of the factors in an orthogonal array, the neighboring points in two dimensions present a complete arrangement. If $rand = 1$, some points will scatter on the edge of the region. However, a random number $rand \in (0, 1]$ is used to make the orthogonal points located in the region with a random distance from the ant.

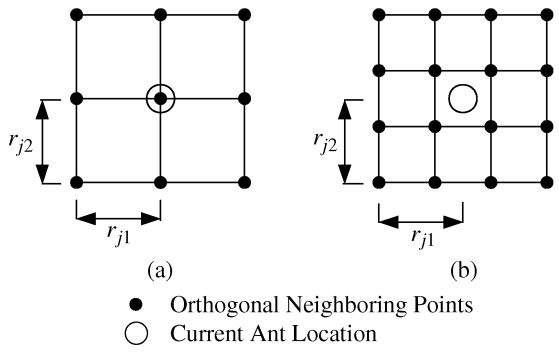


Fig.3. Examples of orthogonal neighboring points in two dimensions using the first two columns of $OA(9, 4, 3)$ and $OA(16, 5, 4)$, $rand = 1$.

In order to accelerate the speed and enhance the accuracy of finding a good point, the radiuses of a region will be changed according to the ants' performance. More details are given in Section 5.

Once a better point is found, the point will become a new center of the region. The orthogonal exploration for an ant in a region can be outlined as follows.

Step 1: Choose a region by (1).

Step 2: Randomly choose n different columns of the given orthogonal array $OA(N, k, s)$ as a new orthogonal array.

Step 3: Generate N neighboring points by (3) and (4).
Step 4: Adaptively adjust the radiuses of the region.
Step 5: Move the region center to the best point.

All m ants perform the above steps and then a globally best point will be found. If the best point ever been found is not changed for σ iterations, this point will be discarded and replaced by a randomly new point. The parameter σ is predefined and this step is to add diversity to the algorithm and avoid trapping in local optima.

4.2 Global Modulation

After all m ants have finished the orthogonal exploration, the domain has been explored for m times. The properties of the regions have been changed and the desirability of every region can be evaluated.

A parameter $\psi \in [0, 1]$ termed elitist rate is used. There are at most $\psi \times \mu$ different regions that will be reserved in the next iteration. The set of the regions is initially denoted as S_R. The best region in S_R is selected and is given a number of rank starting from one. Then the selected region is moved to the set S'_R, which is an elitist set. Initially, $S'_R = \emptyset$. The pheromone τ_j on the selected region j is changed by

$$\tau_j \leftarrow (1-\alpha)\cdot\tau_j + \alpha \cdot T_0 \cdot (\psi\cdot\mu + 1 - rank_j + visit_j) \quad (5)$$

where $\alpha \in (0, 1)$ is the pheromone decay parameter and T_0 is the initial pheromone value. $visit_j$ is the number of ants having visited the region. The better the region and the more times it was visited by ants, the more pheromone is deposited in the region.

After the above procedure, the regions left in S_R are replaced by randomly generated regions and the pheromone in these regions are reset to T_0. Then the reserved regions stored in S'_R are moved back to the new set S_R.

The global modulation can be outlined as follows.

Step 1. Set the variable $ranking = 1$. $S'_R = \emptyset$.
Step 2. Find the best region j in S_R.
Step 3. Set $rank_j = ranking$ and update the pheromone value of region j by (5). Move region j into S'_R.
Step 4. Update $ranking = ranking + 1$.
Step 5. If $ranking \geqslant \psi \times \mu$, goto Step 6. Else goto Step 2.
Step 6. Randomly generate regions to replace the regions left in S_R. Move all regions in S'_R into the new S_R.

4.3 Implementation of COAC

The main steps in continuous orthogonal ant colony (COAC) algorithm are the orthogonal exploration and the global modulation. An overall flowchart of COAC is illustrated in Fig.4, where *MAXITER* is the predefined maximum number of the iteration number. A complete set of pseudocode of the COAC algorithm for a minimization problem is listed in Appendix A.

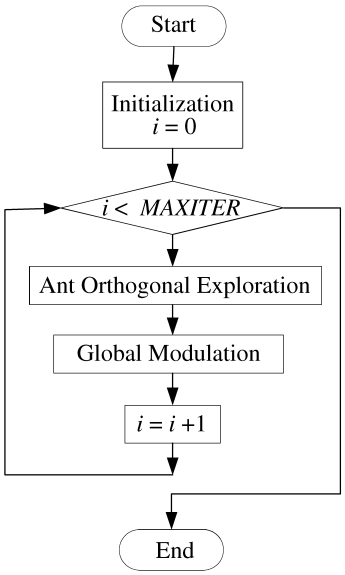


Fig.4. Flowchart of the continuous orthogonal ant colony (COAC).

5 Additional Features

In order to enhance the flexibility of the algorithm, an additional feature, "Adaptive Regional Radius", is introduced to the proposed algorithm.

The search radiuses of a region are able to adaptively shrink and expand according to whether a better region point has been found or not. At first, μ regions are randomly generated within the given domain. The initial radius r_{ji} $(i = 1, 2, \ldots, n)$ for a region j is set as a random value in $(0, u_i - l_i]$, where l_i is the lower bound and u_i is the upper bound of the variable. If a better point is found by the orthogonal exploration, the radiuses of the region will be expanded in order to cover a wider range of neighboring points for more diversity. If no better points are found by the orthogonal exploration, the radiuses of the region will be shrunk to achieve a higher sensitivity by investigating its smaller neighborhood.

The radius shrinking equation for a region j is

$$r_{ji} \leftarrow r_{ji} \cdot shrink, \quad i = 1, \ldots, n \quad (6)$$

where *shrink* is a constant number in (0, 1].

When the value of *shrink* in (6) becomes $1/shrink$

Table 3. List of Test Functions

Test Functions	Domain	Optimum
$f_1 = \sum_{i=1}^{n} x_i^2$	$[-100, 100]^n$	0
$f_2 = \sum_{i=1}^{n} \lvert x_i \rvert + \prod_{i}^{n} \lvert x_i \rvert$	$[-10, 10]^n$	0
$f_3 = \sum_{i=1}^{n} \left(\sum_{j=1}^{i} x_j \right)^2$	$[-100, 100]^n$	0
$f_4 = \sum_{i=1}^{n-1} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$	$[-100, 100]^n$	0
$f_5 = \sum_{i=1}^{n} (\lfloor x_i + 0.5 \rfloor)^2$	$[-100, 100]^n$	0
$f_6 = \sum_{i=1}^{n} i x_i^4$	$[-1.28, 1.28]^n$	0
$f_7 = \sum_{i=1}^{n} -x_i \sin(\sqrt{x_i})$	$[-500, 500]^n$	-1675.93^*
$f_8 = \sum_{i=1}^{n} [x_i^2 - 10\cos(2\pi x_i) + 10]$	$[-5.12, 5.12]^n$	0
$f_9 = -20\exp(-0.2\sqrt{1/n \sum_{i=1}^{n} x_i^2}) - \exp\left(1/n \sum_{i=1}^{n} \cos 2\pi x_i\right) + 20 + e$	$[-32, 32]^n$	0
$f_{10} = \pi/n \Big\{ 10\sin^2(3\pi y_1) + \sum_{i=1}^{n-1} (y_i - 1)^2 [1 + \sin^2(3\pi y_{i+1})] + (y_n - 1)^2 [1 + \sin^2(2\pi x_n)] \Big\} + \sum_{i=1}^{n} u(x_i, 5, 100, 4)^{**}$	$[-50, 50]^n$	0
$f_{11} = 1/10 \Big\{ \sin^2(3\pi x_1) + \sum_{i=1}^{n-1} (x_i - 1)^2 [1 + \sin^2(3\pi x_{i+1})] + (x_n - 1)^2 [1 + \sin^2(2\pi x_n)] \Big\} + \sum_{i=1}^{n} u(x_i, 10, 100, 4)^{**}$	$[-50, 50]^n$	0
$f_{12} = \sum_{i=1}^{11} [a_i - x_i(b_i^2 + b_i x_i)/(b_i^2 + b_i x_3 + x_4)]^{2**}$	$[-5, 5]^n$	$3.075 \times 10^{-4*}$
$f_{13} = -\sum_{i=1}^{5} [(x - a_i)(x - a_i)^{\mathrm{T}} + c_i]^{-1**}$	$[0, 10]^n$	-10.1532^*
$f_{14} = -\sum_{i=1}^{7} [(x - a_i)(x - a_i)^{\mathrm{T}} + c_i]^{-1**}$	$[0, 10]^n$	-10.4029^*
$f_{15} = -\sum_{i=1}^{10} [(x - a_i)(x - a_i)^{\mathrm{T}} + c_i]^{-1**}$	$[0, 10]^n$	-10.5364^*
$f_{16} = \sum_{i=1}^{n} [y_i^2 - 10\cos(2\pi y_i) + 10]$, $\boldsymbol{y}^{\mathrm{T}} = \boldsymbol{M} \cdot \boldsymbol{x}^{\mathrm{T}}$	$[-5.12, 5.12]^n$	0
$f_{17} = -20\exp\left(-0.2\sqrt{1/n \sum_{i=1}^{n} y_i^2}\right) - \exp\left(1/n \sum_{i=1}^{n} \cos 2\pi y_i\right) + 20 + e$, $\boldsymbol{y}^{\mathrm{T}} = \boldsymbol{M} \cdot \boldsymbol{x}^{\mathrm{T}}$	$[-32, 32]^n$	0

Note: n = dimension of the problem, Optimum = the known best value

*The accurate minimum values of these functions are not known. The values shown above are approximate values.

**Detailed descriptions of these functions are given in Appendix B.

as in

$$r_{ji} \leftarrow r_{ji}/shrink, \quad i = 1, \ldots, n \tag{7}$$

the radius is expanded. Note that the radius may be shrunk too small to take any effect. Therefore, when the radius is smaller than a lower bound φ, the radius of the i-th variable is reset as a random value within $[\varphi, u_i - l_i]$.

6 Experimental Discussions

The algorithm proposed in this paper is designed for solving continuous optimization problems. The most common form of continuous optimization is function optimization. In this section, seventeen continuous functions are tested.

6.1 Problem Definitions

The test functions are all minimization problems with the form

$$\begin{aligned} &\text{minimize} \quad f(\boldsymbol{x}), \\ &\text{subject to} \quad \boldsymbol{l} \leqslant \boldsymbol{x} \leqslant \boldsymbol{u}, \end{aligned}$$

where $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$, n is the dimensions of the problem. $\boldsymbol{l} = (l_1, l_2, \ldots, l_n)$ and $\boldsymbol{u} = (u_1, u_2, \ldots, u_n)$ are the lower bounds and the upper bounds of the corresponding variables in $\boldsymbol{x}$, which define the feasible domain of the problem.

The seventeen test functions are listed in Table 3, where f_1 to f_6 are unimodal functions, f_7 to f_{15} are multimodal functions with some local optima, f_{16} and f_{17} are rotated multimodal functions. The rotated matrix $\boldsymbol{M}$ is generated by Salomon's method in [38].

6.2 Comparisons Between COAC and Other Ant Algorithms

In this subsection, the performance of our proposed continuous orthogonal ant colony (COAC) algorithm will be compared with two other ant algorithms, namely, API[15] and continuous ACO (CACO)[13]. The dimensions for the functions are all set as four.

Based on the difficulties of the functions, some basic parameters in COAC are set separately to different functions as in Table 4. These basic parameters include the number of regions μ, the number of ants m and the maximum function evaluations $MAXEVALS$ in each trial. The value of q_0 used in the orthog-

onal exploration and the shrinking ratio *shrink* are also set. The value of *ERROR* is used to measure the fact that whether the optimization is successful. If the errors between the result and the known optimum is smaller than or equal to *ERROR* (*ERROR* $\leqslant$ $|result - optimum|$), this trial is regarded as successful.

Table 4. Basic Parameter Settings for Test Functions in COAC

F	μ	m	*MAXEVALS*	q_0	*shrink*	*ERROR*
f_1	30	20	170 000	0.5	0.3	0.1
f_2	30	20	170 000	0.5	0.3	0.1
f_3	30	20	170 000	0.5	0.5	0.1
f_4	50	50	400 000	0.5	0.7	1.0
f_5	30	20	170 000	0.5	0.2	0.1
f_6	30	20	170 000	0.5	0.4	0.1
f_7	200	100	900 000	0.3	0.7	1.0
f_8	200	100	900 000	0.3	0.8	0.1
f_9	200	100	900 000	0.3	0.4	0.1
f_{10}	50	50	400 000	0.3	0.8	0.1
f_{11}	50	50	400 000	0.3	0.8	0.1
f_{12}	50	50	1 000 000	0.3	0.9	0.0001
f_{13}	30	20	170 000	0.3	0.3	0.1
f_{14}	30	20	170 000	0.3	0.3	0.1
f_{15}	30	20	170 000	0.3	0.8	0.1
f_{16}	200	100	900 000	0.3	0.8	0.1
f_{17}	200	100	900 000	0.3	0.4	0.1

The values of α, T_0, the elitist rate ψ in (5) and the parameters σ and φ are set as 0.1, 0.0001, 10%, 20, 9.99×10^{-324} respectively. The number of levels used is set as $s = 3$. Each function is tested for 100 independent trials with the same parameter settings.

The parameter settings in API are: the number of ants $m = 20$; the default number of sites memorized in each ant's memory $p = 2$; the number of explorations performed by each ant between two nest moves $T = 50$. Please refer to [15] for more details about API. The parameter settings in CACO are: the number of regions $\mu = 200$, the number of local ants $L = 10$, the number of global ants $G = 90$. Please refer to [13] for more details about CACO. The maximum function evaluations before termination are the same as that of COAC.

In Tables 5 and 6, computational data of API, CACO and COAC are listed. The accuracy comparisons in Table 5 are made by the mean values (mean) and the standard deviation (dev) of the functions. The performances between COAC and API, COAC and CACO are judged by using a two-tailed test. Compared with the other two algorithms, in most cases the enhancements by COAC are significant. It can be seen that the proposed COAC can solve unimodal functions with higher accuracy. Although API and CACO achieve better mean values than COAC in f_8, the differences are not significant. To most of the multimodal functions, COAC can find out near optimum

Table 5. Accuracy Comparisons Between API, CACO, and COAC

F	Optimum	API		CACO		COAC		COAC-API	COAC-CACO
		mean	dev	mean	dev	mean	dev	t-test	t-test
f_1	0	1.04×10^{-3}	5.59×10^{-4}	8.77×10^{-35}	4.09×10^{-34}	0	0	$-18.6572^{\dagger}$	$-2.1451^{\dagger}$
f_2	0	6.35×10^{-3}	1.70×10^{-3}	3.85×10^{-21}	2.36×10^{-20}	0	0	$-37.2742^{\dagger}$	$-1.6340^{\dagger}$
f_3	0	2.34×10^{-3}	1.34×10^{-3}	6.66×10^{-23}	1.49×10^{-22}	0	0	$-17.4707^{\dagger}$	$-4.4602^{\dagger}$
f_4	0	4.91×10^{-1}	2.54×10^{0}	7.35×10^{-1}	1.97×10^{0}	1.98×10^{-1}	5.44×10^{-1}	-1.1275	$-2.6229^{\dagger}$
f_5	0	0	0	0	0	0	0	0	0
f_6	0	5.18×10^{-14}	5.40×10^{-14}	2.89×10^{-75}	1.19×10^{-74}	0	0	$-9.5841^{\dagger}$	$-2.4464^{\dagger}$
f_7	-1675.93^{*}	-1427.21	119.63	-1675.93	4.34×10^{-12}	-1675.93	4.32×10^{-12}	$-20.7912^{\dagger}$	0
f_8	0	2.49×10^{-4}	1.39×10^{-4}	9.95×10^{-3}	9.95×10^{-2}	1.99×10^{-2}	1.3×10^{-1}	1.4036	0.5793
f_9	0	1.44×10^{-2}	4.61×10^{-3}	6.95×10^{-16}	6.09×10^{-16}	5.89×10^{-16}	0	$-31.1873^{\dagger}$	-1.7498
f_{10}	0	1.27×10^{-4}	7.42×10^{-5}	1.18×10^{-31}	1.32×10^{-46}	1.18×10^{-31}	1.32×10^{-46}	$-17.1474^{\dagger}$	0
f_{11}	0	2.67×10^{-4}	1.58×10^{-4}	1.35×10^{-32}	2.48×10^{-47}	1.35×10^{-32}	2.48×10^{-47}	$-16.8937^{\dagger}$	0
f_{12}	$3.075 \times 10^{-4*}$	4.09×10^{-3}	7.67×10^{-3}	4.49×10^{-4}	1.16×10^{-4}	3.09×10^{-4}	8.73×10^{-6}	$-4.9356^{\dagger}$	$-12.0371^{\dagger}$
f_{13}	-10.1532^{*}	-8.7365	2.2828	-7.8333	3.4229	-10.1532	1.86×10^{-14}	$-6.2061^{\dagger}$	$-6.7796^{\dagger}$
f_{14}	-10.4029^{*}	-9.3475	2.1210	-9.4478	2.4887	-10.4029	1.52×10^{-14}	$-4.9762^{\dagger}$	$-3.8381^{\dagger}$
f_{15}	-10.5364^{*}	-9.4104	2.1944	-10.2491	1.4180	-10.5364	1.43×10^{-14}	$-5.1314^{\dagger}$	$-2.0263^{\dagger}$
f_{16}	0	2.61×10^{-4}	1.24×10^{-4}	1.08	8.50×10^{-1}	2.06×10^{-1}	4.27×10^{-1}	$4.8259^{\dagger}$	$-9.2332^{\dagger}$
f_{17}	0	1.41×10^{-2}	4.01×10^{-3}	6.60×10^{-16}	5.00×10^{-16}	5.89×10^{-16}	0	$-35.2578^{\dagger}$	-1.4214

†The value of t with 198 degrees of freedom is significant at $a = 0.05$ by a two-tailed test.

*The accurate minimum values of these functions are not known. The values shown above are approximate values.

Table 6. Speed Comparisons Between API, CACO and COAC

F	API			CACO			COAC		
	evals	err_evals	%ok	evals	err_evals	%ok	evals	err_evals	%ok
f_1	88 754	4 202	100	169 669	2 151	100	44 076	915	100
f_2	85 613	5 679	100	169 460	1 507	100	94 953	968	100
f_3	85 655	6 157	100	169 820	3 705	100	52 473	1 061	100
f_4	253 490	57 883	96	399 148	14 365	88	395 808	46 086	97
f_5	3 641	3 641	100	1 537	1 537	100	743	743	100
f_6	92 550	1 561	100	167 731	203	100	26 498	147	100
f_7	417 581	5 547	6	69 202	1 940	100	264 230	158 730	100
f_8	507 354	60 570	100	50 235	2 200	99	137 395	84 991	98
f_9	428 845	8 206	100	227 616	2 633	100	58 522	10 901	100
f_{10}	191 756	5 929	100	109 224	1 619	100	37 697	5 418	100
f_{11}	202 759	3 651	100	120 924	1 744	100	39 637	1 850	100
f_{12}	585 770	105 426	62	957 071	280 064	44	582 348	187 214	100
f_{13}	100 652	13 939	72	27 904	6 860	68	46 209	7 510	100
f_{14}	93 497	12 936	80	45 205	3 530	87	49 252	7 339	100
f_{15}	96 741	13 376	79	36 680	3 700	96	45 774	8 798	100
f_{16}	477 496	46 333	100	48 705	7 978	25	298 129	268 555	80
f_{17}	471 245	8 113	100	204 240	2 767	100	47 409	9 078	100

results in all trials. COAC can also solve rotated multimodal functions. Note that the mean value by API in f_{16} is better than COAC and CACO for it has a higher success rate, as shown in Table 6. COAC is better than CACO and API in solving f_{17}. The convergent curves of the COAC, CACO and API of some selected functions are drawn in Fig.5 to illustrate the average trends and the best trends in the optimization process.

In Table 6, speed comparisons are made. The column "evals" stands for the average function evaluations in the trials for obtaining the result. The column "err_evals" stands for the average function evaluations when the successful results within predefined errors listed in Table 4 have been obtained. The column "%ok" stands for the success rate within the errors. The average number of function evaluations (evals) in COAC is smaller than that in CACO except for f_7, f_8, f_{13}, f_{14}, f_{15}, f_{16} and is smaller than that in API except for f_2 and f_4. The proposed COAC needs more function evaluations than CACO in solving f_7, f_8, because it takes time to locate the globally best region. Although COAC uses more evaluations than CACO in f_{13}, f_{14}, f_{15}, its success rates are much higher. To f_{16}, API is the best, COAC is the second and CACO succeeds in only 25%. The average function evaluations within the errors (err_evals) reflect the convergent speed when a near best value is obtained. COAC shows a dominate advantage in solving unimodal functions fast and accurately. Although COAC is slower to converge than API and CACO to multimodal functions in the early function evaluations, COAC can always approach to the optimum with high accuracy and its success rates are higher than API and CACO. Fig.6 demonstrates the convergent speed between COAC, CACO and API within the defined errors. The x-coordinates in these figures stand for function evaluations, while the y-coordinates stand for the accumulated times for the success counts in the 100 trials within the errors.

6.3 Analysis of the Radiuses in the Orthogonal Exploration

Fig.7 illustrates the changing process of radiuses compared with the resulting function value in one trial. The radius of the first variable of the best region is drawn in each graph. Although the initial radiuses of each variable are different, their changing processes are similar. So the curves in the graphs can represent the basic situations of all the radiuses. It can be seen that the radiuses are closely related to the resulting function values. The behavior of the adaptively changing radiuses can be concluded as follows.

1) The frequency of shrinking the radiuses is higher than that of expanding them, so that the radiuses tend to shrink with time. The reason is that better points are harder to find as the optimization goes on, so the radiuses shrunk to hold onto the optimal niches.

2) Except the shrinking of the radiuses, there are three cases that the radius of the best region is expanded. (a) It happens when the best region in this iteration is the one from the previous iteration and a better point is found by the orthogonal exploration. The expansion of radius can help reduce the shrinking

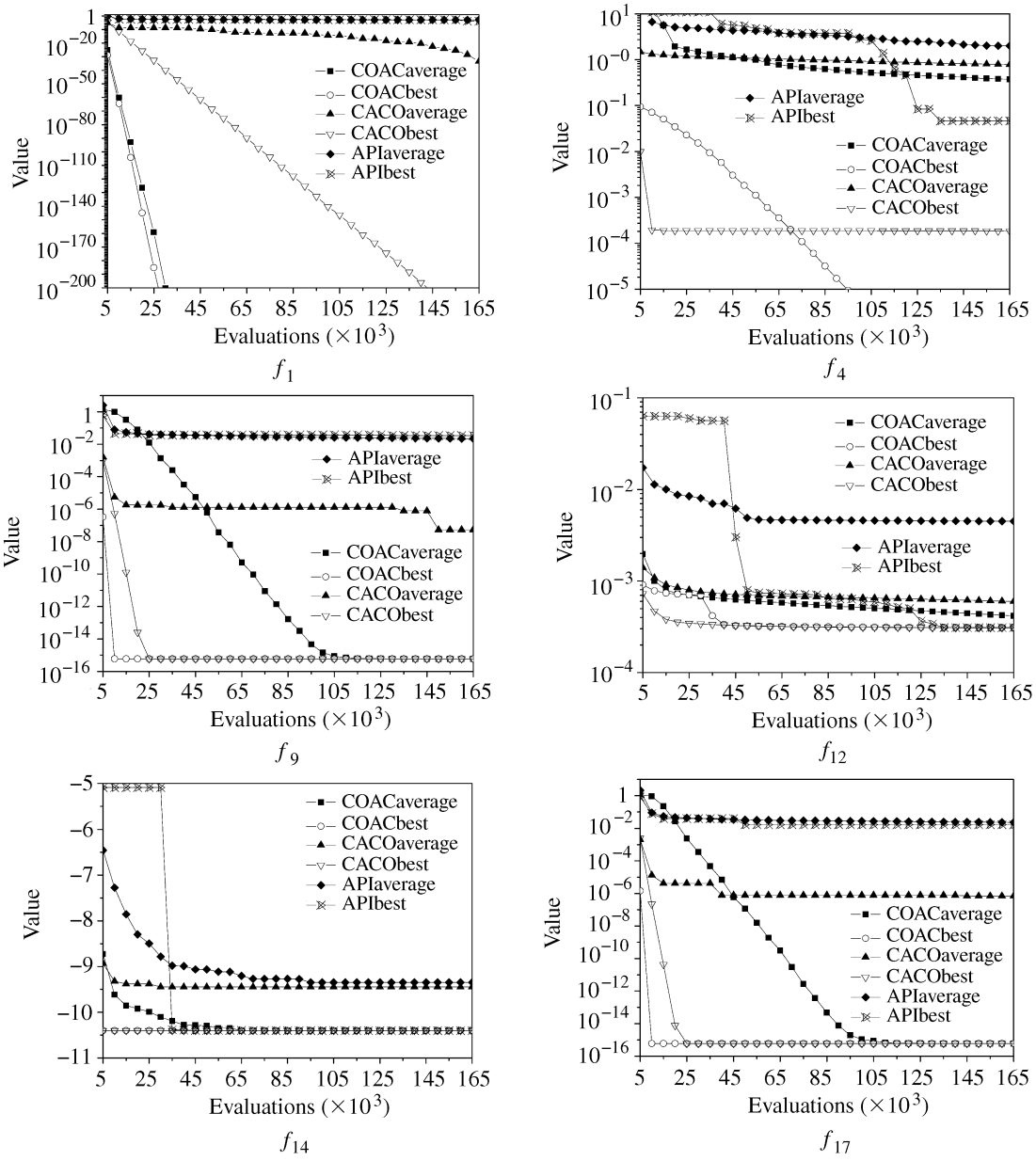
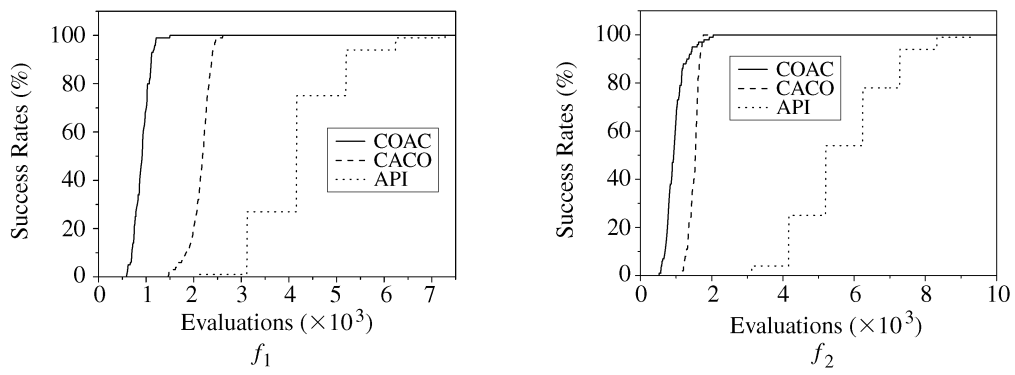
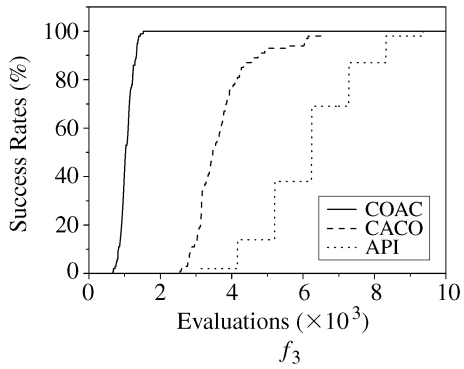


Fig.5. Convergent curves comparing the scalability of COAC, CACO and API for selected functions.

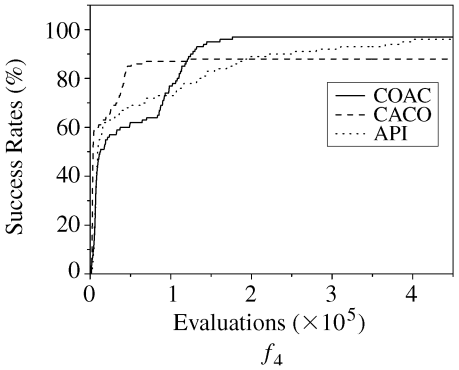


To be continued

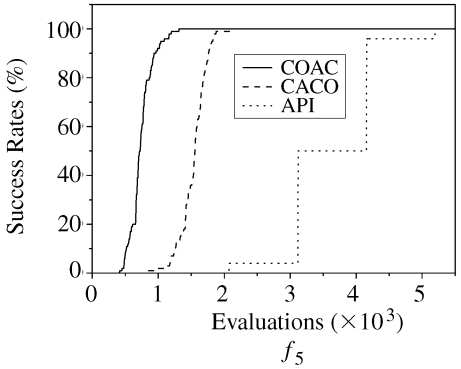
Continue from the previous page



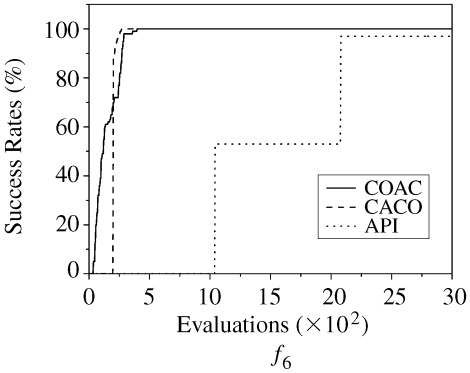
f_3



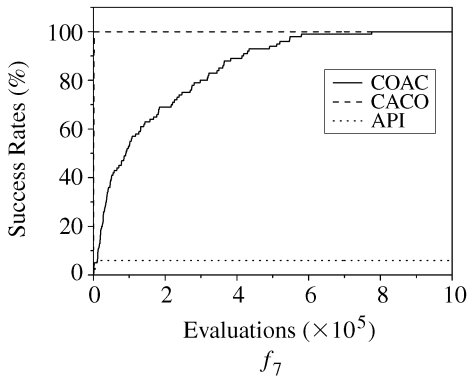
f_4



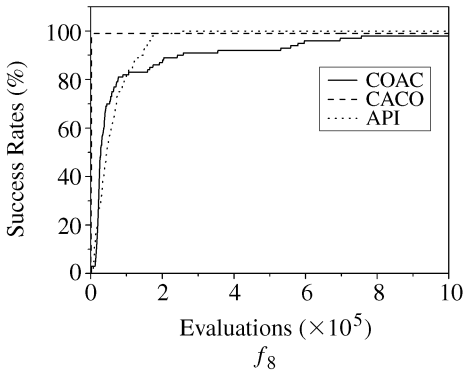
f_5



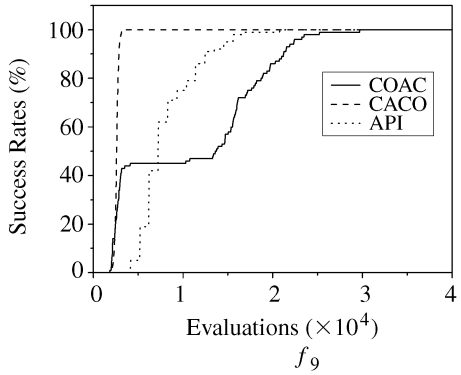
f_6



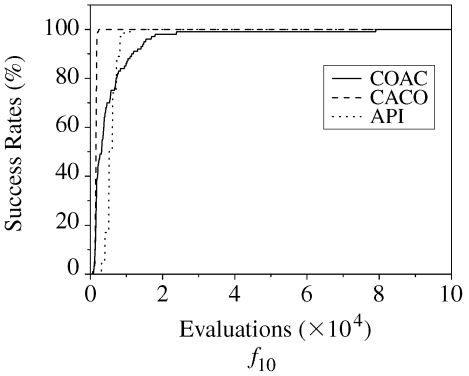
f_7



f_8



f_9



f_{10}

To be continued

Continue from the previous page

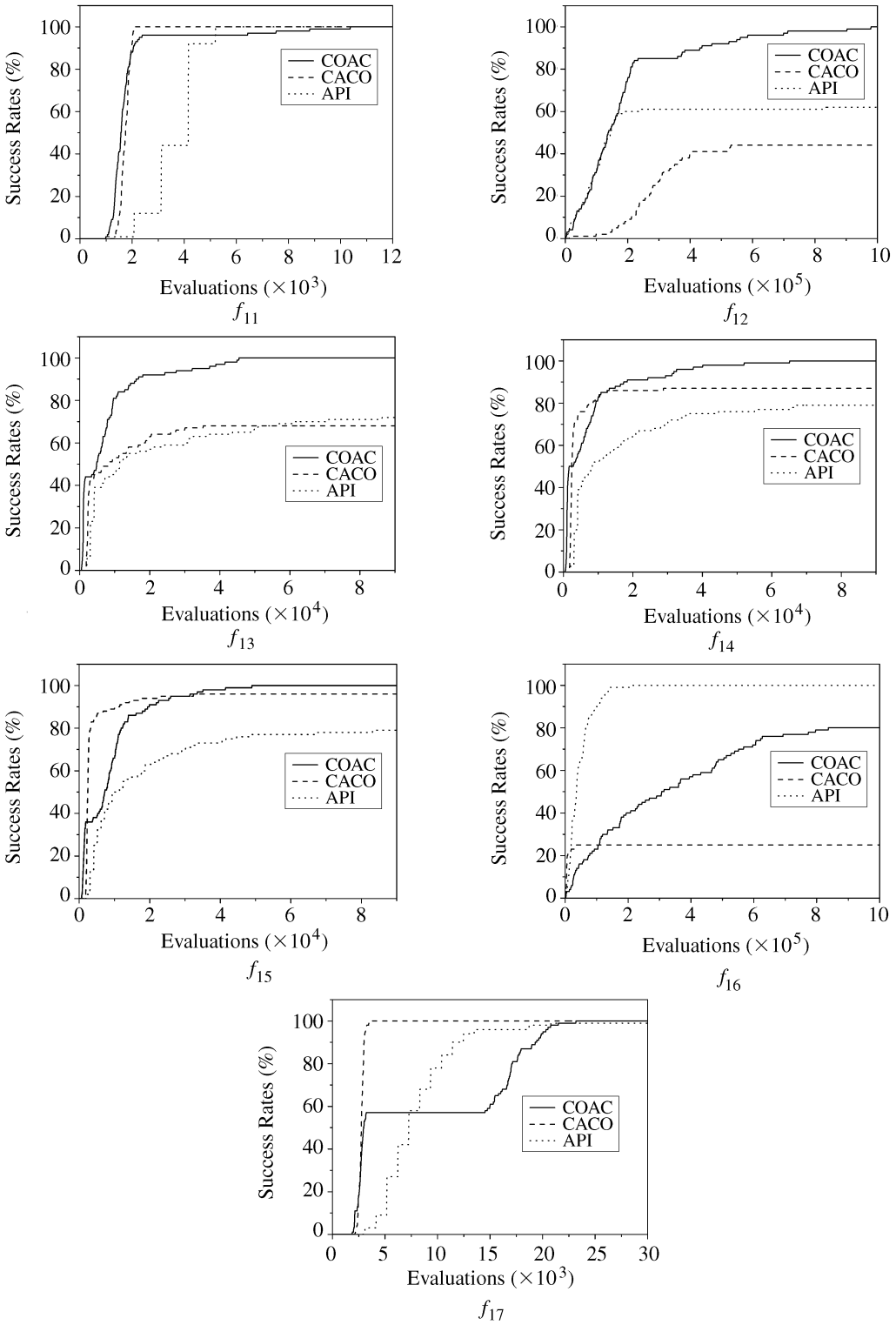
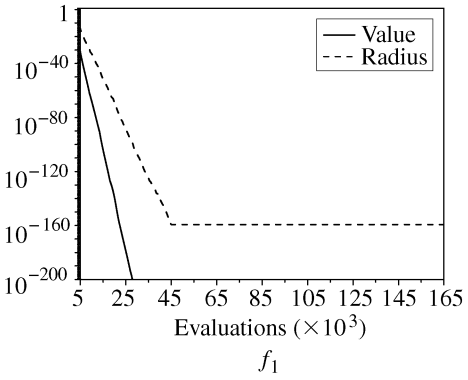
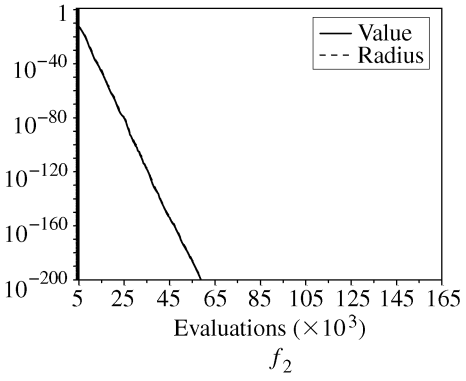


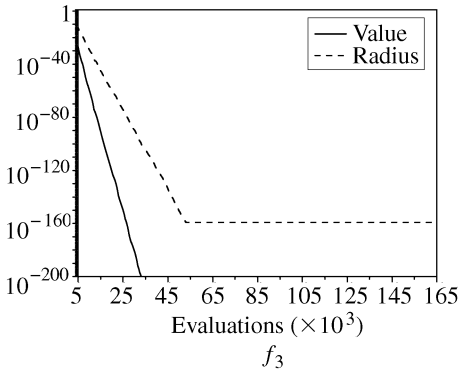
Fig.6. Success rates of results within *ERROR* in Table 4.



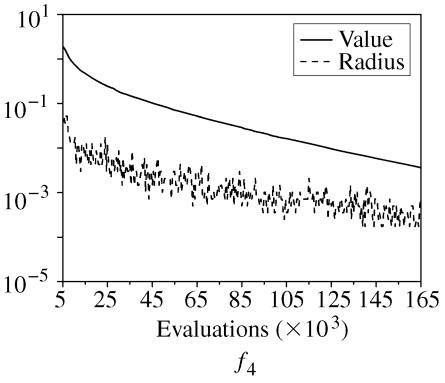
f_1



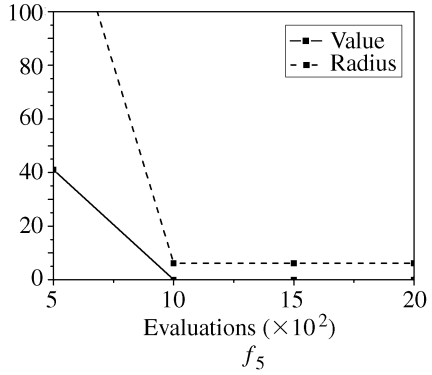
f_2



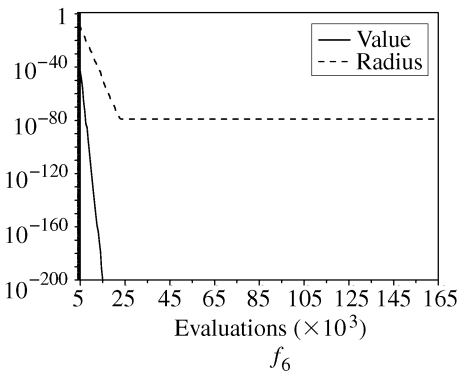
f_3



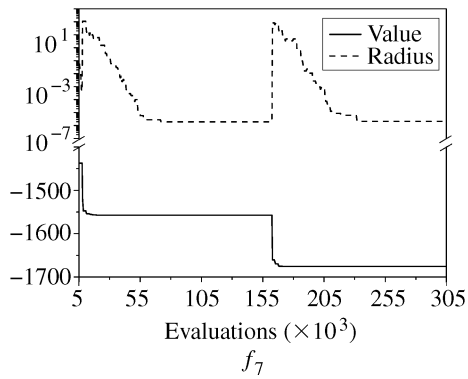
f_4



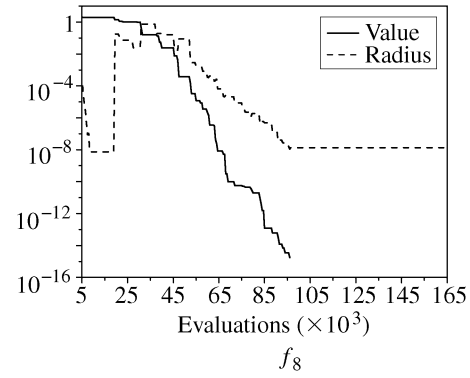
f_5



f_6



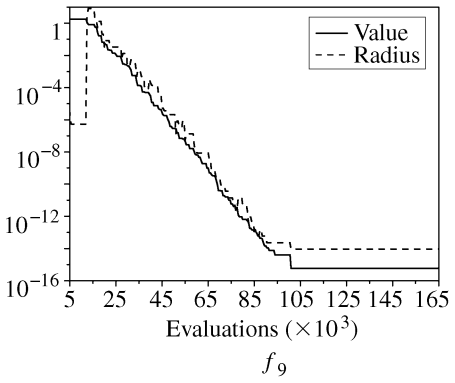
f_7



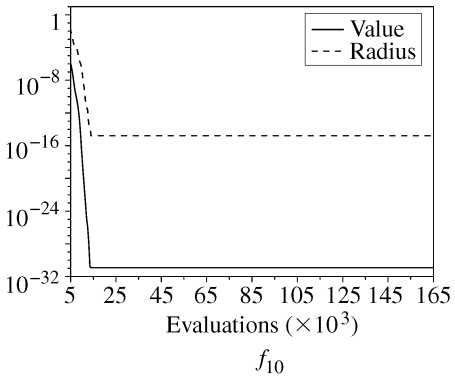
f_8

To be continued

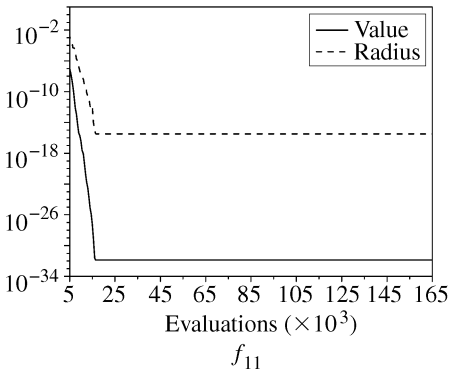
Continue from the previous page



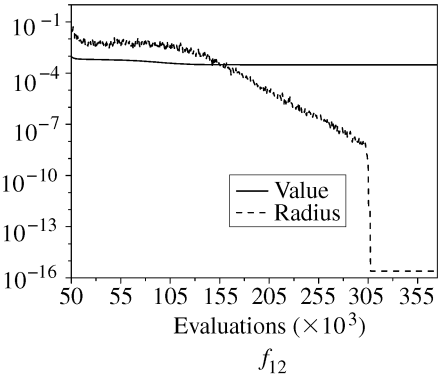
f_9



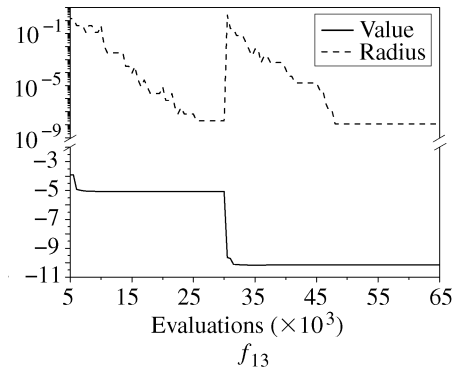
f_{10}



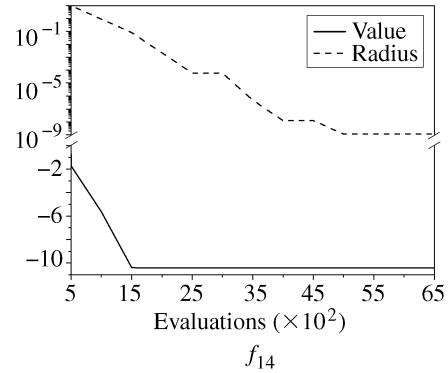
f_{11}



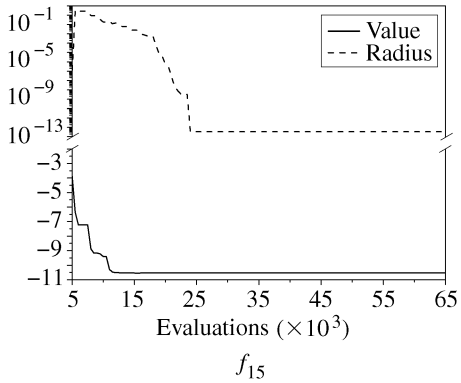
f_{12}



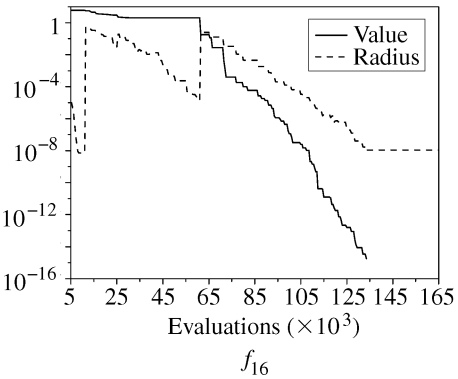
f_{13}



f_{14}



f_{15}



f_{16}

To be continued

Continue from the previous page

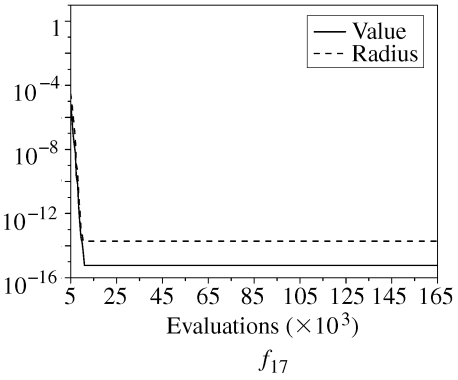


Fig.7. Function values and radius values by the COAC in one trial.

speed and avoid trapping in local optima and converge prematurely. (b) The best region becomes a region with a larger radius in the current iteration. (c) If the radius is smaller than the predefined value, it will be reset to a random value that may be larger than the previous one.

For unimodal functions, the radiuses are gradually reduced except for f_4 in Fig.7. For multimodal functions and f_4, whose global optima are harder to find, the radiuses fluctuate with step changes. Each step jump accompanies a great reduction of the function values, where large radiuses can help jump out of local optima.

3) As the graphs are drawn by radiuses and function values of the best results, the radiuses are unchanged if no better results have been found. Hence, horizon lines are shown in the graph, such as the radiuses of f_1 and f_3 etc.

It can be seen from the graphs that the sizes of the radiuses control the accuracy of the resulting function value. The smaller the radiuses, the higher the accuracy of the result is achieved.

7 Conclusions

This paper has developed a novel algorithm, continuous orthogonal ant colony (COAC), to solve continuous optimization problems. It utilizes the orthogonal design method for ant colony optimization (ACO) to search the continuous domain completely and effectively.

The performance of the proposed COAC algorithm has been compared with that of two other ant algorithms, API and CACO, in solving seventeen continuous functions. The results show that the proposed COAC algorithm is among the fastest and the most accurate in solving unimodal functions. Although COAC converges slower at an early stage than API and CACO in solving some multimodal functions, in most cases it can find global optimum values with higher accuracy and higher success rates.

Moreover, the orthogonal design method and the adaptive radius adjustment method present great potentials to the optimization field. These methods will be extended to other relevant algorithms for delivering wider advantages in solving those problems. Currently, research is carried out to make the radius changing criterion more sensible to the dynamical optimization process.

References

[1] Deneubourg J L, Aron S, Goss S, Pasteels J M. The self-organizing exploratory pattern of the Argentine ant. *Journal of Insect Behavior*, 1990, 3: 159–168.

[2] Goss S, Aron S, Deneubourg J L, Pasteels J M. Self-organized shortcuts in the Argentine ant. *Naturwissenschaften*, 1989, 76(12): 579–581.

[3] Dorigo M, Stützle T. Ant Colony Optimization. the MIT Press, 2003.

[4] Dorigo M, Gambardella L M. Ant colony system: A cooperative learning approach to the traveling salesman problem. *IEEE. Trans. Evol. Comput.*, 1997, 1(1): 53–66.

[5] Toth P, Vigo D. The Vehicle Routing Problem. SIAM Monographs on Discrete Mathematics and Applications, Philadelphia, Society for Industrial & Applied Mathematics, 2001.

[6] Gambardella L M, Taillard É D, Agazzi G. MACS-VRPTW: A Multiple Ant Colony System for Vehicle Routing Problems with Time Windows. New Ideas in Optimization, Corne D, Dorigo M, Glover F (eds.), London, McGraw Hill, 1999, pp.63–76.

[7] Zhang J, Hu X M, Tan X, Zhong J H, Huang Q. Implementation of an ant colony optimization technique for job shop scheduling problem. *Transactions of the Institute of Measurement and Control*, 2006, 28(1): 1–16.

[8] Zecchin A C, Simpson A R, Maier H R, Nixon J B. Parametric study for an ant algorithm applied to water distribution system optimization. *IEEE Trans. Evol. Comput.*, 2005, 9: 175–191.

[9] Parpinelli R S, Lopes H S, Freitas A A. Data mining with an ant colony optimization algorithm. *IEEE Trans. Evol. Comput.*, 2002, 4: 321–332.

[10] Sim K M, Sun W H. Ant colony optimization for routing and load-balancing: Survey and new directions. *IEEE Trans. Systems, Man, and Cybernetics — Part A: System and Humans*, 2003, 33: 560–572.

[11] Bilchev G, Parmee I C. The ant colony metaphor for searching continuous design spaces. In *Proc. the AISB Workshop on Evolutionary Computation*, University of Sheffield, UK, *LNCS* 933, Springer-Verlag, Berlin, Germany, 1995, pp.25–39.

[12] Wodrich M, Bilchev G. Cooperative distributed search: The ant's way. *Control and Cybernetics*, 1997, 3: 413–446.

[13] Mathur M, Karale S B, Priye S, Jyaraman V K, Kulkarni B D. Ant colony approach to continuous function optimization. *Ind. Eng. Chem. Res.*, 2000, 39: 3814–3822.

[14] Holland J H. Adaptation in Natural and Artificial Systems. Second Edition (First Edition, 1975), Cambridge: the MIT Press, MA, 1992.

[15] Monmarché N, Venturini G, Slimane M. On how *Pachycondyla apicalis* ants suggest a new search algorithm. *Future Generation Computer Systems*, 2000, 16: 937–946.
[16] Dréo J, Siarry P. Continuous interacting ant colony algorithm based on dense heterarchy. *Future Generation Computer Systems*, 2004, 20: 841–856.
[17] Dréo J, Siarry P. A new ant colony algorithm using the heterarchical concept aimed at optimization of multiminima continuous functions. In *Proc. ANTS* 2002, Brussels, Belgium, *LNCS* 2463, 2002, pp.216–221.
[18] Socha K. ACO for continuous and mixed-variable optimization. In *Proc. ANTS* 2004, Brussels, Belgium, *LNCS* 3172, 2004, pp.25–36.
[19] Socha K, Dorigo M. Ant colony optimization for continuous domains. *Eur. J. Oper. Res.*, 2008, 185(3): 1155–1173.
[20] Pourtakdoust S H, Nobahari H. An extension of ant colony system to continuous optimization problems. In *Proc. ANTS* 2004, Brussels, Belgium, *LNCS* 3172, 2004, pp.294–301.
[21] Kong M, Tian P. A binary ant colony optimization for the unconstrained function optimization problem. In *Proc. International Conference on Computational Intelligence and Security* (*CIS'05*), Xi'an, China, *LNAI* 3801, 2005, pp.682–687.
[22] Kong M, Tian P. A direct application of ant colony optimization to function optimization problem in continuous domain. In *Proc. ANTS* 2006, Brussels, Belgium, *LNCS* 4150, 2006, pp.324–331.
[23] Chen L, Shen J, Qin L, Chen H J. An improved ant colony algorithm in continuous optimization. *Journal of Systems Science and Systems Engineering*, 2003, 12(2): 224–235.
[24] Dréo J, Siarry P. An ant colony algorithm aimed at dynamic continuous optimization. *Appl. Math. Comput.*, 2006, 181: 457–467.
[25] Feng Y J, Feng Z R. An immunity-based ant system for continuous space multi-modal function optimization. In *Proc. the Third International Conference on Machine Learning and Cybernetics*, Shanghai, August 26–29, 2004, pp.1050–1054.
[26] Shelokar P S, Siarry P, Jayaraman V K, Kulkarni B D. Particle swarm and ant colony algorithms hybridized for improved continuous optimization. *Appl. Math. Comput.*, 2006, doi: 10.1016/j.amc. 2006.09.098.
[27] Rao C R. Factorial experiments derivable from combinatorial arrangements of arrays. *J. Royal Statist. Soc.*, 1947, 9(Suppl.): 128–139.
[28] Bush K A. Orthogonal arrays [Dissertation]. University of North Carolina, Chapel Hill, 1950.
[29] Math Stat Res Group, Chinese Acad Sci. Orthogonal Design. Bejing: People Education Pub., 1975. (in Chinese)
[30] Fang K T, Wang Y. Number-Theoretic Methods in Statistics. New York: Chapman & Hall, 1994.
[31] Hedayat A S, Sloane N J A, Stufken J. Orthogonal Arrays: Theory and Applications. New York: Springer-Verlag, 1999.
[32] Nathanson M B. Elementary Methods in Number Theory. New York: Springer-Verlag, 2000.
[33] Zhang Q, Leung Y W. An orthogonal genetic algorithm for multimedia multicast routing. *IEEE Trans. Evolutionary Computation*, 1999, 3(1): 53–62.
[34] Leung Y W, Wang W. An orthogonal genetic algorithm with quantization for global numerical optimization. *IEEE Trans. Evol. Comput.*, 2001, 5(1): 41–53.
[35] Ho S Y, Chen J H. A genetic-based systematic reasoning approach for solving traveling salesman problems using an orthogonal array crossover. In *Proc. the Fourth Internal Conference/Exhibition on High Performance Computing in the Asia-Pacific Region*, May 2000, 2: 659–663.
[36] Liang X B. Orthogonal designs with maximal rates. *IEEE Trans. Information Theory*, 2003, 49(10): 2468–2503.
[37] Tanaka H. Simple genetic algorithm started by orthogonal design of experiments. In *Proc. SICE Annual Conference in Sapporp*, August 2004, pp.1075–1078.
[38] Salomon R. Reevaluating genetic algorithm performance under coordinate rotation of benchmark functions. *BioSystems*, 1996, 39: 263–278.

Xiao-Min Hu received her BSc. degree in computer science in 2006 from Sun Yat-Sen University, Guangzhou, China. She is currently a Ph.D. candidate majored in computer application and technology in Sun Yat-Sen University. Her research interests include artificial intelligence, evolutionary computation, routing optimization and biological information.

Jun Zhang received the Ph.D. degree in electrical engineering from City University of Hong Kong, in 2002. From 2003 to 2004, he was a Brain Korean 21 Postdoctoral Fellow in the Department of EECS, Korea Advanced Institute of Science and Technology (KAIST). Since 2004, he has been with the Sun Yat-Sen University, Guangzhou, China, where he is currently a professor with the Department of Computer Science. He has authored three research book chapters and over 50 refereed technical papers in his research areas. His research interests include genetic algorithms, ant colony system, fuzzy logic, neural network, cash flow optimization and nonlinear time series analysis and prediction.

Yun Li received the B.S. degree in radio electronics science from Sichuan University, Chengdu, China, in 1984, an M.Sc. degree in electronic engineering from University of Electronic Science and Technology of China (UESTC), Chengdu, in 1987, and a Ph.D. degree in computing and control engineering from University of Strathclyde, Glasgow, U.K., in 1990. From 1989 to 1990, he worked at the U.K. National Engineering Laboratory and for Industrial Systems and Control Limited, Glasgow, U.K. He became a lecturer at the University of Glasgow in 1991. In 2002, he served as a visiting professor at Kumamoto University, Japan. He is currently a senior lecturer at University of Glasgow and a visiting professor at UESTC. In 1996, he independently invented the "indefinite scatter-

ing matrix" theory, which opened up a ground-breaking way for microwave feedback circuit design. From 1987 to 1991, he carried out leading work in parallel processing for recursive filtering and feedback control. In 1992, he achieved first symbolic computing for power electronic circuit design, without needing to invert any matrix, complex-numbered or not. Since 1992, he has pioneered into design automation of control systems and discovery of novel systems using evolutionary learning and intelligent search techniques. He established the IEEE CACSD Evolutionary Computation Working Group and the European Network of Excellence in Evolutionary Computing (EvoNet) Workgroup on Systems, Control, and Drives in 1998. He has supervised 12 Ph.D.s in this area and has over 140 publications. Dr. Li is a chartered engineer, a member of the Institution of Engineering and Technology, and a member of the IEEE.

Appendix A COAC ALGORITHM

1) /* Initialization phase */
   For each region j **do**
      CreateRegion(j)
   End-for
   best := MAXVALUE
   localBest := MAXVALUE
   countE := 0
2) /* Orthogonal exploration phase */
   For each region j **do** visit$_j$:= 0 **End-for**
   For k := 1 to m **do**
      Choose the next region j according to (1) (2)
      visit$_j$:= visit$_j$ + 1
      OrthogonalExplore(j) /* Do orthogonal exploration to region j */
      If (Function(j) < *localBest*) **then**
        localBest := Function(j)
        If (*localBest* < *best*) **then**
          best := *localBest*
        End-if
      End-if
   End-for
   If (*lastBest* = *localBest*) **then**
      countE := *countE* + 1
   Else
      lastBest := *localBest*
      countE := 0
   End-if
   If (*countE* > σ) **then**
      CreateRegion(*localbestRegion*)
      localBest := MAXVALUE
   End-if
3) /* Global modulation phase */
   countG := 0
   $S_{jR} := \emptyset$
   $S'_{jR} = \emptyset$
   For j := 1 to μ **do**
      rank$_j$:= 0
      Add region j to S_{jR}
   End-for
   For i :=1 to $\psi \times \mu$ **do**
      Find the region j with the minimum value satisfied $j \in S_{jR}$
      Update the pheromone in region j according to (5)
      Move region j to S'_{jR} /* Region j is deleted in S_{jR}*/
   End-for
   For all region $j \in S_{jR}$ **do**
      CreateRegion(j)
   End-for
   Move all regions in S'_{jR} to S_{jR}
4) **If** (End_condition = True) **then** print best
   Else goto Phase 2)

Appendix B Detailed Description of Some Test Functions

1) f_{10} and f_{11}

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

$$u(x_i, a, p, j) = \begin{cases} p(x_i - a)^j, & x_i > a, \\ 0, & -a \leqslant x_i \leqslant a, \\ p(-x_i - a)^j, & x_i < -a, \end{cases}$$

2) f_{12}

$$f_{12} = \sum_{i=1}^{11} \left[a_i - \frac{x_1(b_i^2 + b_i x_2)}{b_i^2 + b_i x_3 + x_4} \right]^2,$$

a_i and b_i^{-1} are as follows:

i	1	2	3	4	5	6
a_i	0.1957	0.1947	0.1735	0.1600	0.0844	0.0627
b_i^{-1}	0.25	0.5	1	2	4	6
i	7	8	9	10	11	
a_i	0.0456	0.0342	0.0323	0.0235	0.0246	
b_i^{-1}	8	10	12	14	16	

3) $f_{13} \sim f_{15}$

$$f(x) = -\sum_{i=1}^{p} [(x - a_i)(x - a_i)^{\mathrm{T}} + c_i]^{-1}$$

where a_i is as follows:

i	a_{ij}, $j = 1, 2, \ldots, 4$				c_i
1	4	4	4	4	0.1
2	1	1	1	1	0.2
3	8	8	8	8	0.2
4	6	6	6	6	0.4
5	3	7	3	7	0.4
6	2	9	2	9	0.6
7	5	5	3	3	0.3
8	8	1	8	1	0.7
9	6	2	6	2	0.5
10	7	3.6	7	3.6	0.5

The value of p is equal to 5, 7, and 10 respectively from $f_{13} \sim f_{15}$.