A New Differential Evolution with Self-terminating Ability using Fuzzy Control and K-Nearest Neighbors

J.C.Y. LAI, *Member, IEEE*, F.H.F. Leung, *Senior Member, IEEE* and S.H. Ling, *Member, IEEE*

***Abstract*—A new Differential Evolution (DE) that incorporates fuzzy control and k-nearest neighbors algorithm to determine the terminating condition is proposed. A technique called Iteration Windows is introduced to govern the number of iteration in each searching stage. The size of the iteration windows is controlled by a fuzzy controller, which uses the information provided by the k-nearest neighbors system to analyze the population during the searching process. The controller keeps controlling the iteration windows until the end of the searching process. The wavelet based mutation process is embedded in the DE searching process to enhance the searching performance of DE. The *F* weight of DE is also controlled by the fuzzy controller to further speed up the searching process. A suite of benchmark test functions is employed to evaluate the performance of the proposed method. It is shown empirically that the proposed method can terminate the searching process with a reasonable number of iteration.**

I. Introduction

Differential Evolution (DE) is a population based stochastic optimization algorithm that searches the solution space to find out the solution. DE has been well accepted as a powerful algorithm for handling optimization problems during the last decade [1]. It uses the weighted difference between two population vectors to determine a third vector. No separate probability distribution has to be used so that the scheme is completely self-organizing [2]. It is a new member to the class of Evolutionary Algorithms (EA) that imitates the process of biological evolution. Owing to the population based strategy, EAs are less possibly getting trapped in local optima. As a result, many researchers view EAs as global optimization algorithms. Other important examples of EAs include the Genetic Algorithm (GA) [3] and Evolutionary Programming (EP) [4].

Similar to GA, DE guides the population towards the global solution within the given solution space by using evolutionary operations. Comparing with other optimization algorithms, DE is relatively easy to implement. It requires fewer parameters for tuning, and has a relatively fast convergence speed. A simple vector subtraction is able to generate a random direction of movement for the population to search the solution space. DE can also offer a high degree of exploration for the population. It has been successfully applied in many optimization problems such as data clustering [5], power plant control [6], optimization of non-linear functions [7], etc. However, on applying DE to real problems, the users usually need to input the maximum number of iteration as the stopping criterion of the algorithm. The maximum number of iteration depends on the problem nature. As there is not any strong proof for the best number of iteration, the user can only determine this number by trial and error and/or the user's experience. Determining the maximum number of iteration is not an easy task in many problems. To tackle this problem, a technique called Iteration Windows is proposed for embedding into the DE searching process in this paper.

On doing the searching process in DE, the movement of the population takes an important role in finding the global optimal point. The distribution of the population in the search domain provides much information about the searching process. By capturing the information of the population, we can understand the state of the searching process. Some example states could be: searching evenly in different areas of the domain; population moving toward a region in the domain; part of the population being trapped in some local optimal point, etc.

Clustering is one of the methods to obtain the population information in the searching process. In this paper, an improved DE that incorporates fuzzy control and k-nearest neighbors to determinate the terminating condition is proposed. Based on this method, the searching process is divided into different stages with different number of iteration. Each stage is characterized by an iteration window. The population distribution of the pervious stage will be analyzed by the k-nearest neighbor algorithm. The fuzzy controller will employ the result of the k-nearest neighbor algorithm to determine the next-stage iteration window's size. With the evolution of the population along the searching process, the size of the iteration windows will be decreasing until it becomes zero. Then, the searching process ends.

This paper is organized as follows: Section II presents the operation of the DE with self-terminating ability. Experimental study and analysis are given in Section III. Benchmark test functions are used to evaluate the performance of the proposed method. A conclusion will be drawn in Section IV.

II. DE with Self-terminating ability

DE is an optimization method gaining its importance in the field of evolutionary computation techniques. A randomly generated population over the solution space will first be obtained. The population of solution vectors is then

J.C.Y. Lai is with the Centre for Signal Processing, Dept. of Electronic and Information Engg., The Hong Kong Polytechnic University, Hung Ham, Hong Kong. (e-mail: 08900438r@polyu.edu.hk).

F.H.F. Leung is with the Centre for Signal Processing, Dept. of Electronic and Information Engg., The Hong Kong Polytechnic University, Hung Ham, Hong Kong. (e-mail: enfrank@inet.polyu.edu.hk).

S.H. Ling is with the Centre for Health Technologies, Faculty of Engineering and Information Technology, University of Technology, Sydney, NSW, Australia. (e-mail: steve.ling@uts.edu.au).

successively updated by addition, subtraction and component swapping; until the population converges to the optimum after going through a given number of iteration. The pseudo code for the standard DE (SDE) process is shown in Fig.1. In this paper, a DE with self-terminating ability using K-Nearest Neighbors and Fuzzy Control is proposed. The pseudo code is shown in Fig. 2. The details of both the SDE and the proposed DE with self-terminating ability are discussed as follows.

A. Standard Differential Evolution (SDE)

DE attempts to maintain a population of N_p vectors for each generation of evolution, with each vector contains D elements of parameters. Let $P_{x,g}$ be the population of the current generation g, and $\mathbf{x}_{i,g}$ be the i-th vector in this population:

$$\begin{aligned} P_{x,g} &= (\mathbf{x}_{i,g}), i = 0,1,...,N_p - 1; g = 0,1,...,g_{\max} \\ \mathbf{x}_{i,g} &= (x_{j,i,g}), j = 0,1,...,D-1. \end{aligned} \tag{1}$$

where $g_{\max}$ is the maximum generation number. Before the population can be initialized over the solution space, the boundary of the searching space should be specified. The population should be uniformly and randomly distributed in the searching space. Once initialized, DE creates a mutated vector, $\mathbf{v}_{i,g}$ for each target vector $\mathbf{x}_{i,g}$ by using the mutation operation. In particular, DE adds a scaled, randomly sampled, vector difference to form a third vector. The mutated vector is realized by the following equation:

$$\mathbf{v}_{i,g} = \mathbf{x}_{i,g} + F \cdot (\mathbf{x}_{r_1,g} - \mathbf{x}_{r_2,g}) \tag{2}$$

where F is the scaling factor; r_1 and r_2 are two different integers which are randomly generated from $\{0, 1, ..., N_p-1\}$. To complement the differential mutation search strategy and increase the diversity of the perturbed parameter vectors, DE employs a method called uniform crossover. Each vector element pair $x_{j,i,g}$ and $v_{j,i,g}$ generates a new trial vector element $u_{j,i,g}$. The crossover operation is realized by the following equation:

$$\mathbf{u}_{i,g} = (u_{j,i,g}) = \begin{cases} (v_{j,i,g}) & \text{if } (rand_j(0,1) \le Cr \\ (x_{j,i,g}) & \text{otherwise.} \end{cases} \tag{3}$$

where $Cr \in [0, 1]$ is called the crossover rate, which is a user-defined value that controls the fraction of parameters that are copied from the mutant. $rand_j(0,1)$ generates a random value between 0 and 1 for the j-th parameter. The algorithm also ensures $u_{j,i,g}$ gets at least one parameter value as $x_{j,i,g}$ [1]. Then the population is updated by comparing each trial vector to the corresponding target vector $\mathbf{x}_{i,g}$. If the fitness function value of the trial vector is lower than that of the target vector, replace the target vector in the next generation; otherwise the target vector retains its place in the population for at least one generation. The selection operation is therefore realized by the following equation:

$$\mathbf{x}_{i,g+1} = \begin{cases} \mathbf{u}_{i,g} & \text{if } f(\mathbf{u}_{i,g}) \le f(\mathbf{x}_{i,g}) \\ \mathbf{x}_{i,g} & \text{otherwise.} \end{cases} \tag{4}$$

where $f(\cdot)$ is the fitness function. Because of this selection operation, DE is expected to have high optimization ability. When the condition to stop further evolution is satisfied, for example, the preset maximum number of iteration has been reached, the algorithm ends with the best solution as the final solution (see Fig. 1).

```
begin
   Initialize the population;
   While (not the maximum number of iteration) do
      {
         Mutation operation by equation (2);
         Crossover operation by equation (3);
         Evaluation of the fitness function;
         Select the best vector by equation (4);
      }
end
```

Fig. 1. Pseudo code for SDE.

```
begin
Initialize the population;
   While (control not equal to 0) do
   {
      While (Iteration window not equal 0) do
      {
         Mutation operation by equation (2);
         Crossover operation by equation (3);
         Evaluation of the fitness function;
         Modifying the trial population vectors by
         equation (5);
         Select the best vector by equation (4);
         Iteration window =Iteration window – 1;
      }
      Analyze the population by K-NN by equation (10);
      Count the number of local clusters;
      Determine the new $k$ , the new $F$ and iteration
      window size by the fuzzy controller;
      If iteration window > 6 then control = 1;
      If iteration window < = 6 then control = 0;
   }
end
```

Fig. 2. The operation of the proposed system.

C. Wavelet mutation in crossover operation

The crossover operation of (3) is done with respect to the elements of the trial vector (after mutation) in DE. We can embed a wavelet mutation in the DE crossover operation. It exhibits a fine-tuning ability [18]. The details are as follows. The crossover after the first mutation takes place according to (3). Let $\mathbf{u}_{i,g} = (u_{0,i,g}, u_{1,i,g}, ..., u_{D-1,i,g})$ be the i-th vector after crossover for the wavelet mutation. Its element value is inside the vector element's boundary [$para_{\min}^j, para_{\max}^j$]. The mutated crossover vector is given by

$\bar{\mathbf{u}}_{i,g} = \left(\bar{u}_{0,i,g}, \bar{u}_{1,i,g}, \ldots, \bar{u}_{D-1,i,g}\right)$, and

$$\bar{u}_{j,i,g} = \begin{cases} u_{j,i,g} + \sigma \times \left(para^{j}_{\max} - u_{j,i,g}\right) \text{ if } \sigma > 0 \\ u_{j,i,g} + \sigma \times \left(u_{j,i,g} - para^{j}_{\min}\right) \text{ if } \sigma \le 0 \end{cases}, \tag{5}$$

$$\sigma = \frac{1}{\sqrt{a}} e^{-\left(\frac{\varphi}{a}\right)^2 / 2} \cos\left(5\left(\frac{\varphi}{a}\right)\right), \tag{6}$$

where

$$a = e^{-\ln(\lambda) \times \left(1 - \frac{t}{T}\right)^{\zeta_{wm}} + \ln(\lambda)} \tag{7}$$

A larger value of $|\sigma|$ at the early stage of evolution gives a larger searching space for the solution; when $|\sigma|$ is small at the later stage of evolution, the algorithm gives a smaller searching space for fine-tuning.

After the operation of the wavelet mutation, the population is updated by comparing each trial vector $\bar{\mathbf{u}}_{i,g}$ to the corresponding target vector $\mathbf{x}_{i,g}$ using the method of standard DE as given by (4). A new population is generated and the same evolution process is repeated. Such an iterative process will be terminated when a defined number of iteration has been met.

D. K-Nearest Neighbors (K-NN)

The k-nearest neighbor algorithm (K-NN) is a method for classifying data element based on closest training data samples in a multidimensional feature space, where each training data sample is associated with a class label [8][9][10]. A data element is classified by counting the majority class of its neighbors. The number of training data samples being considered is controlled by the value of *k*, which is a positive integer given by the user. In most of the general applications, the value of *k* is small. For example, if *k* = 1, the data element is assigned to the same class of its nearest neighbor.

In this paper, the *k*-NN algorithm is used to identify the number of groups of elements (clusters) in the searching domain. A user-defined threshold *T* is introduced. This is a user-defined value for controlling the size of the cluster. Firstly, the entire population is evaluated by the *k*-NN algorithm, where *k* is a value controlled by a fuzzy controller. Afterward, the *k* nearest elements would be determined by *k*-NN. The total distance between those nearest elements and the current test element would be calculated by the following equation:

$$D_{total} = \sum_{i=1}^{k} d_i(\mathbf{p}, \mathbf{q}_i) \tag{8}$$

where $d_i(\cdot)$ is the Euclidean distance between two points; $\mathbf{p}$ is the current testing point and $\mathbf{q}_i$ is one of the *k* nearest elements in the searching domain. The Euclidean distance is defined as follows.

$$\begin{aligned} d_i(\mathbf{p}, \mathbf{q}_i) &= \sqrt{(p_1 - q_{i1})^2 + (p_2 - q_{i2})^2 + \cdots + (p_n - q_{in})^2} \\ &= \sqrt{\sum_{j=1}^{n} (p_j - q_{ij})^2} \end{aligned} \tag{9}$$

The average distance d_{average} of the group (cluster) is evaluated by the following equation:

$$d_{average} = \frac{D_{total}}{k} \tag{10}$$

If d_{average} is smaller than the threshold *T*, a local cluster is obtained. If d_{average} is larger than the threshold *T*, a cluster is not present. The system will count the number of clusters obtained. Then, this information is passed to the fuzzy controller to determine the size of the next iteration window. The detail of the operation of the iteration window will be discussed in the next section. The main purpose of applying K-NN is to analyze the population distribution information of the evolution.

E. Iteration Windows

At present, many optimization algorithms require the user to input the maximum number of iteration as the terminating criterion. This number is often determined by trial and error and/or the user's experience. We propose a technique called the Iteration Windows to determine the number of iteration by a fuzzy controller.

The optimization algorithm will first run for a small number of iteration, and then the evolution will be paused. This small number of iteration forms the stage-1 iteration window (Fig. 3). After going through this iteration window, the population will be saved. The population distribution information (i.e. the number of local clusters) obtained from *k*-NN will be recorded and passed to the fuzzy controller to determine the size of the next-stage iteration window. One advantage of using the iteration window is that it can reduce the computation power for reaching the final solution, as the total number of iteration is governed by the fuzzy controller instead of arbitrarily set by the user.

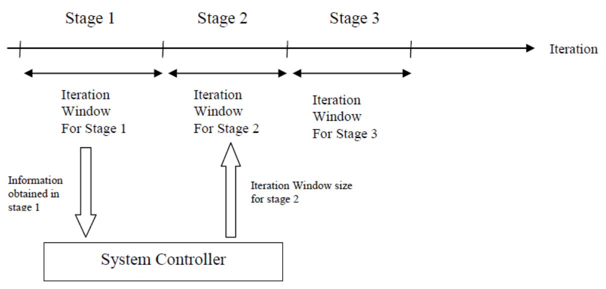
Stage 1
Stage 2
Stage 3
Iteration
Iteration Window For Stage 1
Iteration Window For Stage 2
Iteration Window For Stage 3
Information obtained in stage 1
Iteration Window size for stage 2
System Controller

Fig. 3. The operation of Iteration Windows.

The relation of the next iteration window's size (the next number of iteration) to the population distribution could be described by some heuristic rules. Then, the fuzzy controller is to model the relation. Under this approach, the total number of iteration could be controlled with respect to the nature of the problem. It does not require the user to determine the total number of iteration, but the system has its own ability to stop the system when it has reliably converged to the real optimum point. The next section will discuss how the fuzzy controller controls the size of the iteration window in each stage.

F. Fuzzy Controller

A fuzzy controller realizes the fuzzy inference [11], which is a process of making decisions by using fuzzy logic and fuzzy rules. The use of fuzzy logic in rule-based systems has been a success, with applications in climate control [12], medicine [13], relational database [14] and scheduling [15]. The process of fuzzy inference involves membership functions, logical operations and fuzzy if-then rules. The membership functions enable the description of the inputs and outputs in linguistic terms. The logical operations and fuzzy if-then rules can easily be derived based on human knowledge.

There are two major methods for implementing fuzzy inference systems. They are the Mamdani-type and Sugeno-type [16] methods. The Mamdani-type fuzzy inference method is used in this paper to construct the fuzzy controller. It uses linguistic control rules obtained from experienced human experts to perform decision making. For the Mamdani-type inference system, the output variables must be fuzzy sets. To map the fuzzy set to some crisp output value, the system needs to perform a process called defuzzification, which is the process of generating quantifiable results from fuzzy sets and the corresponding membership functions. In the proposed system, the method of center of gravity (COG) is used to perform defuzzification. Fig. 4 shows how the fuzzy inference system works.

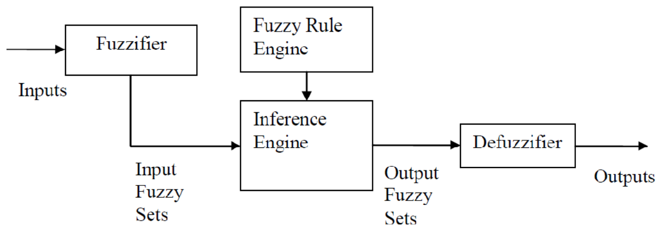

Fig. 4. The operation of fuzzy inference system.

The evolution of population-based searching algorithms, including DE, will not be the same for different experiments of a given problem because the algorithms themselves contain a lot of uncertainties. On analyzing the population distribution in the searching process, the derivation of the next iteration window's size from the population distribution should best be described as some heuristic rules. The aforementioned fuzzy controller can then be employed to determine the window's size together with the value of k for the k-NN in the next stage. The fuzzy controller takes the current k value and the population distribution information (i.e. the number of local clusters obtained) provided by k-NN as inputs, and maps this information into the fuzzy membership functions. The fuzzy controller also controls the value of F in the DE operation to enhance the searching process. Fig. 5 shows the input and output of the proposed fuzzy controller. The inference system makes decision based on the fuzzy rules provided by the user.

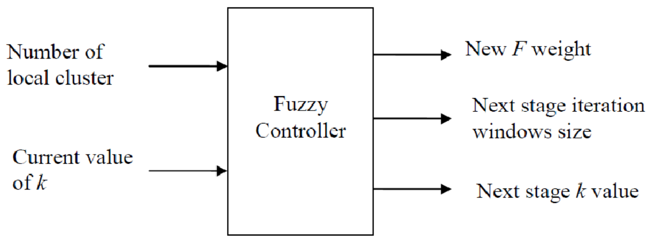

Fig. 5. The input and output of the fuzzy inference system.

Table 1. Benchmark Test Functions.

Test function	**Domain range**	**Optimal point**
Sphere model $f_1(\mathbf{x}) = \sum_{i=1}^{30} x_i^2$	$-50 \le x_i \le 150$	Min(f_1)= f_1([1, ..., 1])=0
Generalized Rosenbrock's function $f_2(\mathbf{x}) = \sum_{i=1}^{29} \left[100 \left(x_{i+1} - x_i^2 \right)^2 + (x_i - 1)^2 \right]$	$-2.048 \le x_i \le 2.048$	Min(f_2)= $f_2(\mathbf{0})$=0
Step function $f_3(\mathbf{x}) = \sum_{i=1}^{30} \left(\lvert x_i + 0.5 \rvert \right)^2$	$-5 \le x_i \le 10$	Min(f_3)= $f_3(\mathbf{0})$=0
Hartman's family 1 $f_4(\mathbf{x}) = -\sum_{i=1}^{4} c_i \exp\left[-\sum_{j=1}^{3} a_{ij} (x_j - p_{ij})^2 \right]$	$0 \le x_i \le 1$	Min(f_4)= f_4([0.114,0.556, 0.833])= −3.8628
Hartman's family 2 $f_5(\mathbf{x}) = -\sum_{i=1}^{4} c_i \exp\left[-\sum_{j=1}^{6} a_{ij} (x_j - p_{ij})^2 \right]$	$0 \le x_i \le 1$	Min(f_5)= f_5([0.201,0.15, 0.477,0.275,0.311,0.627])= −3.32
Generalized Griewank's function $f_6(x) = \frac{1}{4000} \sum_{i=1}^{30} x_i^2 - \prod_{i=1}^{30} \cos\left(\frac{x_i}{\sqrt{i}} \right) + 1$	$-1200 \le x_i \le 600$	Min(f_6)= $f_6(\mathbf{0})$=0
Generalized Ackley's function $f_7(\mathbf{x}) = -20 \exp\left(-0.2 \sqrt{\frac{1}{30} \sum_{i=1}^{30} x_i^2} \right) - \exp\left(\frac{1}{30} \sum_{i=1}^{30} \cos 2\pi x_i \right) + 20 + e$	$-64 \le x_i \le 32$	Min(f_7)= $f_7(\mathbf{0})$=0
Schwefel's function $f_8(x) = \sum_{i=1}^{30} (x_i \sin(\sqrt{\lvert x_i \rvert}))$	$-500 \le x_i \le 500$	Min(f_8)= f_8([420.9687, ..., 420.9687])= −12569.5

III. Benchmark Test Functions and Results

A. Benchmark Test Functions

A suite of eight benchmark test functions [17] are used to test the performance of the proposed system. The details of these functions are shown in Table 1.

Many different kinds of optimization problems are covered by these functions, which can be divided into three categories. The first category covers the unimodal functions f_1, f_2 and f_3 that are symmetric with a single minimum. The second one covers the multimodal functions f_4 and f_5 with only a few local minima. The last one covers the multimodal functions f_6, f_7 and f_8 with many local minima.

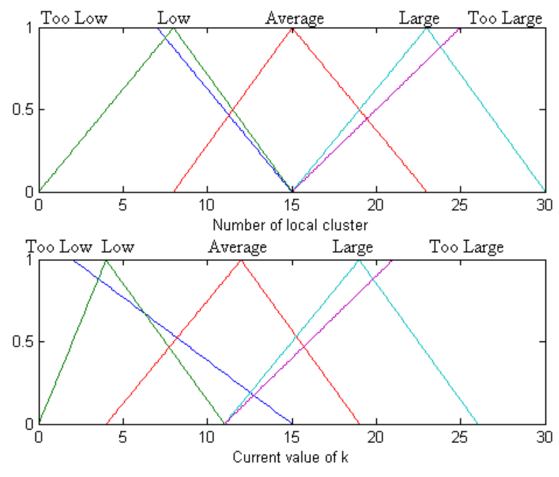

Fig. 6. The output membership functions.

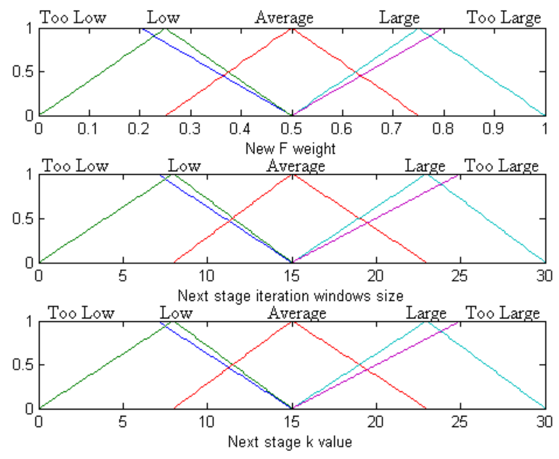

Fig. 7. The input membership functions.

B. *Experimental Setup*

The performance of SDE and the proposed DE with self-terminating ability are evaluated by finding the minimum values of the benchmark test functions. The following simulation conditions are used:

- Initial population: It is generated uniformly at random.
- Crossover Rate: $Cr = 0.5$
- Initial weight factor: $F = 0.7$
- Number of population: $N_p = 30$
- Threshold T for k-NN: 5
- Membership functions: As shown in Fig. 6 and Fig. 7.
- Fuzzy rule: Listed in Table 2.
  (where Input1 is the current number of local clusters; Input2 is the k value of the current iteration window; Output1 is the next F value for DE; Output2 is k for the next stage; Output3 is the next iteration window size.)

Table 2. The fuzzy rule table.

1	If (Input1 is Too Low) then (Output1 is Large).
2	If (Input1 is Low) then (Output1 is Average).
3	If (Input1 is Average) then (Output1 is Low).
4	If (Input1 is Large) then (Output1 is Too Low).
5	If (Input1 is Too Large) then (Output is Too Low).
6	If (Input1 is Too Low) or (Input2 is Too Low) then (Output1 is Large).
7	If (Input1 is Low) or (Input2 is Low) then (Output1 is Large).
8	If (Input1 is Average) or (Input2 is Average) then (Output1 is Average).
9	If (Input1 is Large) or (Input2 is Large) then (Output1 is Too Low).
10	If (Input1 is Too Large) or (Input2 is Too Large) then (Output1 is Too Low).
11	If (Input1 is Too Low) then (Output2 is Too Low).
12	If (Input1 is Low) then (Output2 is Low).
13	If (Input1 is Average) then (Output2 is Average).
14	If (Input1 is Large) then (Output2 is Large).
15	If (Input1 is Too Large) then (Output2 is Too Large).
16	If (Input1 is Too Low) then (Output3 is Too Large).
17	If (Input1 is Low) then (Output3 is Large).
18	If (Input1 is Average) then (Output3 is Average).
19	If (Input1 is Large) then (Output3 is Large).
20	If (Input1 is Too Large) then (Output3 is Too Large).
21	If (Input1 is Too Low) or (Input2 is Too Low) then (Output3 is too Large).
22	If (Input1 is Low) or (Input2 is Low) then (Output3 is Large).
23	If (Input1 is Average) or (Input2 is Average) then (Output3 is Average).
24	If (Input1 is Large) or (Input2 is Large) then (Output3 is Low).
25	If (Input1 is Too Large) or (Input2 is Too Large) then (Output3 is Too Low).

C. *Results and Analysis*

In this section, the simulation results for the 8 benchmark test functions are given to show the merits of the proposed method. The results are shown in Figs. 8 to 15. It is shown that the proposed method can keep the good solution quality and provide higher convergence rate for the 8 benchmark functions when the fuzzy controller in the proposed system is used to terminate the searching process with a reasonable number of iteration and tuning the F value of DE.

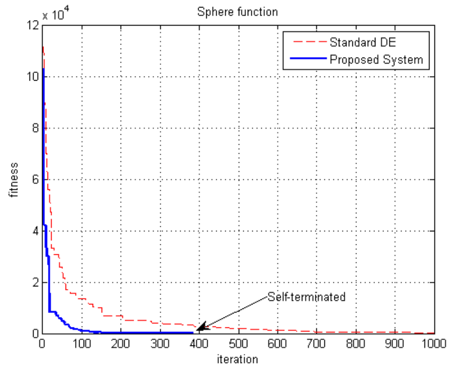

Fig. 8. The sphere function.

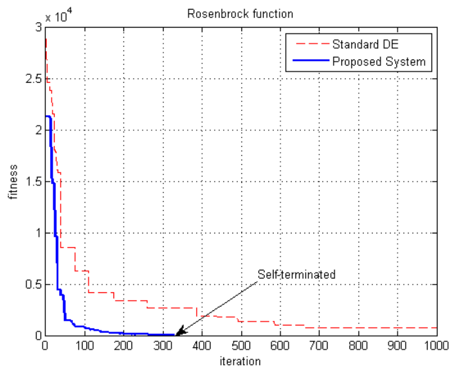

Fig. 9. The Rosenbrock function.

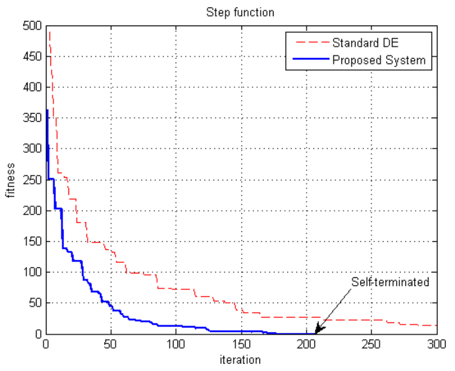

Fig. 10. The step function.

1. Unimodal functions

Function f_1 is a sphere model, which is smooth and symmetric. The main purpose of testing this function is to measure the convergence rate of searching and test the performance of the proposed system on such smooth and symmetric searching condition. It is probably the most widely used function for optimization algorithm testing. For this function, the result is shown in Fig. 8. It shows that the proposed system can terminate the searching process with a reasonable number of iteration. Moreover, because of introducing the wavelet mutation and controlling the value of F in DE, the proposed system can have a higher convergence rate.

Function f_2 is the Rosenbrock function, which is also called the Banana function. The global minimum of these functions is inside a long, narrow, parabolic shaped flat valley. Owing to the smooth and symmetric characteristic of f_2, the main purpose of using this function for testing is to measure the convergence rate of the searching algorithm. The result is shown in Fig. 2. As there is only one minimum within the solution space and the solution space is smooth and symmetric, nearly all the population will move evenly towards that minimum. The proposed method is able to capture the population distribution and terminate the searching process with a reasonable number of iteration. The same as f_1, thanks to introducing the wavelet mutation and controlling the value of F in DE, the proposed system can have a higher convergence rate.

Function f_3 is the step function that is a representation of flat surfaces. Flat surfaces are obstacles for optimization algorithms because they do not give any information about the search direction. Unless the algorithm has a variable step size, it can get stuck in one of the flat surfaces. All methods that involve the mutation operation are good for this function because it can generate a long jump during the evolution. Fig. 10 shows that a higher convergence rate can be obtained by using the proposed system. The same as f_1 and f_2, there is only one minimum within the solution space. Nearly all the population will move evenly towards that minimum. The proposed method is able to capture the population distribution and terminate the searching process with a reasonable number of iteration.

For unimodal functions, the proposed method is able to capture the population distribution and terminate the searching process with a reasonable number. The fuzzy controller captures the population distribution information and generates different size of iteration window. When the population moves toward to the global optimum, the extent of movement of the population will decrease. The fuzzy controller will increase the value of k. The value of k will keep on increasing and the size of the iteration windows will keep on decreasing until it terminates the searching. As a result, the proposed method is able to terminate the searching process with a reasonable number of iteration.

2. Multimodal functions with a few local minima

Two multimodal functions with a few local minima are tested with the proposed method. The results are shown in Fig. 12 and Fig. 13. Function f_4 is the Hartman's family 1 function and function f_5 is the Hartman's family 2 function. Both of them contain some local minima within the searching space. On doing searching, some of the members in the population get trapped in some local minima at the early stage. Since the number of vectors in the local clusters near the local minima is relatively small, the proposed system can detect this situation and drive the DE to keep on performing the searching process. Thanks to the fuzzy control, the proposed system is able to handle this situation and terminate the searching process with a reasonable number of iteration.

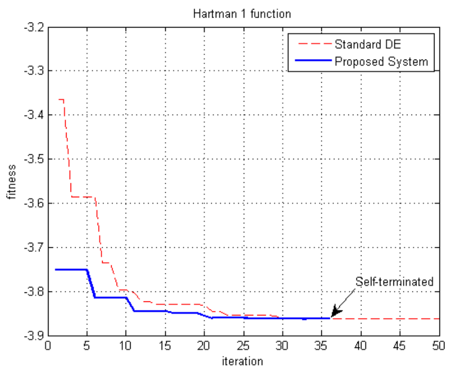

Fig. 11. The Hartman's family 1 function.

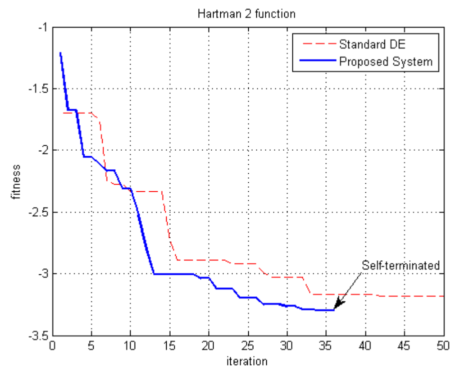

Fig. 12. The Hartman's family 2 function.

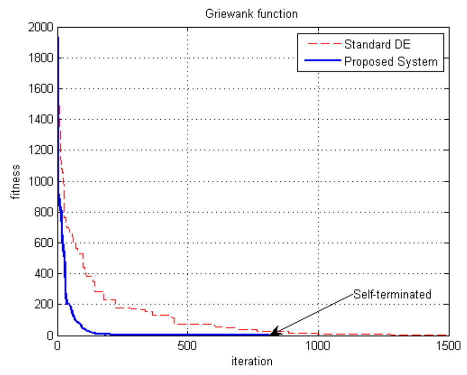

Fig. 13. The Griewank function.

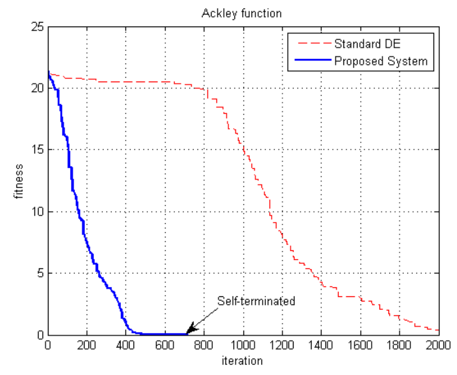

Fig. 14. The Ackley function.

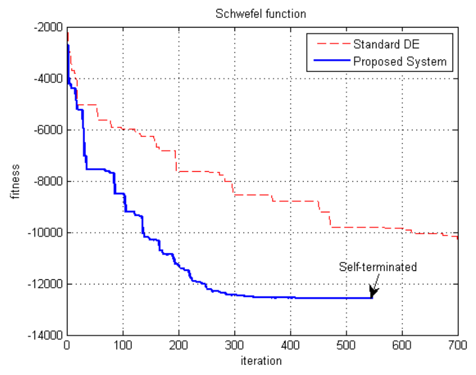

Fig. 15. The Schwefel function.

3. Multimodal functions with many local minima

Functions f_6 is the Generalized Griewank's function which is a multimodal function with many local minima. It has an exponentially increasing number of local minima as its dimension increases and the locations of the minima are regularly distributed. In the experiment, the dimension of the Generalized Griewank's function is 30. So the test function contains plenty of local minima. The test result is shown in Fig. 13. Since there are a lot of local minima, the population can get trapped in them easily. As a result, when k is small (i.e. in the early search stage), there are a lot of local clusters obtained. When the k value increases, the number of obtained local clusters will be decreased. Accordingly, the number of searching stages will increase and a large iteration window (of a maximum value of window size) will also be generated by the fuzzy controller. A large number of iteration will be used by DE to search the optimal point.

Functions f_7 is the Generalized Ackley's function which is a continuous, multimodal function obtained by modulating an exponential function with a cosine wave of moderate amplitude. Its topology is characterized by an almost flat outer region and a central hole or peak where the modulations of the cosine wave become more and more influential. In general, it is a mix of the Easom function and Generalized Griewank's function. The same as f_6, there are a lot of local minima in the searching domain, and the population can get trapped in the local optima easily. When k is small (i.e. in the early search stage), there are a lot of local clusters obtained. Owing to the rapid movement of population towards local optima, the value of k cannot increase rapidly. Moreover, a large iteration window will be generated by the fuzzy controller to handle this situation. When the population moves toward the central area of the search domain where the global optimum is located, the movement of the population vectors will decrease relatively. The fuzzy controller will increase the value of k and fewer local clusters will be obtained. The value of k will keep on increasing and the size of the iteration windows will keep on decreasing until it terminates the searching. The experiment shows that the proposed method can handle such complex situation and terminate the searching process with a reasonable number of iteration without affecting the performance of DE. Moreover, thanks to introducing the wavelet mutation and controlling the value of F in DE, the proposed system can achieve a higher convergence rate.

Functions f_8 is the Schwefel's function which is deceptive in that the global minimum is geometrically distant, over the domain, from the next best local minima. Therefore, the search algorithms are potentially prone to convergence in the wrong direction. Owing to the complexity of the searching domain, DE requires a large number of iteration to perform searching. The experiment shows that the proposed method can handle such complex situation and terminate the searching process with a reasonable number of iteration without affecting the performance of DE. A higher convergence rate can also be obtained thanks to the wavelet mutation and controlling the value of F in DE in the searching process.

For multimodal functions with many local minima, although the searching is a very complex process, the proposed method is able to capture the population distribution information and terminate the searching process with a reasonable number of iteration without affecting the performance of DE. A higher convergence rate can also be obtained when the wavelet mutation and controlling the value of F in DE are employed in the searching process.

IV. Conclusion

In this paper, we proposed a new Differential Evolution (DE) that incorporates fuzzy control and k-nearest neighbor algorithm to determine the terminating condition. A technique called Iteration Windows is introduced to govern the number of iteration in each searching stage. The size of the iteration windows is controlled by a fuzzy controller, which uses the information provided by the k-nearest neighbors to analyze the population in the searching process. The controller keeps controlling the values of k and the size of the iteration windows until the end of the searching process. To enhance the searching performance of DE, the wavelet mutation is employed in the DE operation. Moreover the F weight of DE is also adjusted by the fuzzy controller in the proposed system. A suite of benchmark test functions is employed to evaluate the performance of the proposed method. Experiments show that the proposed method can handle different searching situations and terminate the searching process with a reasonable number of iteration without affecting the performance of DE, and a higher convergence rate can also be obtained.

Acknowledgement

The work described in this paper was substantially supported by a grant from The Hong Kong Polytechnic University (Project No. RP9L).

References

[1] R. Storn and K. Price "Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces," *Journal of Global Optimization*, vol. 11, pp. 341–359, 1997.

[2] U.K. Chakraborty (Ed.), *Advances in Differential Evolution*. Springer, Heidelberg, 2008.

[3] L. Fogel, "Evolutionary programming in perspective: The top-down view," *Computational Intelligence: Imitating Life.* Piscataway, NJ: IEEE Press, 1994.

[4] D. Goldberg, *Genetic Algorithms in Search Optimization and Machine Learning*. Addison-Wesley, 1989.

[5] S. Paterlini and T. Krink, "High performance clustering with differential evolution," in *Proc. IEEE Congress on Evolutionary Computation*, vol. 2, 2004, pp. 2004–2011.

[6] J.H. van Sickel, K.Y. Lee, and J.S. Heo, "Differential evolution and its applications to power plant control," in *Proc. Intelligent Systems Applications to Power Systems 2007*, (ISAP 2007), 5-8 Nov. 2007, pp.1 – 6.

[7] B. Babu and R. Angira, "Optimization of non-linear functions using evolutionary computation," in *Proc. 12th ISME International Conference on Mechanical Engineering*, India, 2001, pp. 153–157.

[8] D. Bremner, E. Demaine, J. Erickson, J. Iacono, S. Langerman, P. Morin, and G. Toussaint, "Output-sensitive algorithms for computing nearest-neighbor decision boundaries," *Discrete and Computational Geometry*, vol. 33, no. 4, 2005, pp. 593-604.

[9] T.M. Cover and P.E. Hart, "Nearest neighbor pattern classification," *IEEE Transactions on Information Theory*, vol. 13, no. 1, 1976, pp. 21-27.

[10] G.T. Toussaint, "Geometric proximity graphs for improving nearest neighbor methods in instance-based learning and data mining," *International Journal of Computational Geometry and Applications*, vol. 15, no. 2, April 2005, pp. 101-150.

[11] G.J. Klir and T.A. Folger, *Fuzzy Sets, Uncertainty, and Information*. Prentice-Hall, 1988.

[12] M.M. Eftekhari and L.D. Marjanovic. "Application of fuzzy control in naturally ventilated buildings for summer conditions," *Energy and Buildings*, vol. 35, issue 7, Aug. 2003.

[13] S. Zahan, C. Micheal, and S. Nikolakeas. *Fuzzy and Neuro-Fuzzy Systems in Medicine, chapter Fuzzy Expert Systems for Myocardial Ischemia Diagnosis*. CRC, 1999.

[14] P. Bosc and O. Pivert, "Extending SQL retrieval feature for the handling of flexible queries," *Fuzzy Information Engineering: A Guided Tour of Applications*, pp 233 -251. Wiley, 1997.

[15] B. Grabot, L. Geneste, and A. Dupeux, "Tuning of fuzzy rules for multiobjective scheduling," *Fuzzy Information Engineering: A Guided Tour of Applications*, pp. 695-703. Wiley, 1997.

[16] Mamdani, E.H. and S. Assilian, "An experiment in linguistic synthesis with a fuzzy logic controller," *International Journal of Man-Machine Studies*, vol. 7, no. 1, pp. 1-13, 1975.

[17] X. Yao and Y. Liu, "Evolutionary programming made faster," *IEEE Trans. Evolutionary Computation*, vol. 3, no. 2, pp. 82-102, July 1999.

[18] S.H. Ling, H.H.C. Iu, K.Y. Chan, H.K. Lam, C.W. Yeung, and F.H.F. Leung, "Hybrid particle swarm optimization with wavelet mutation and its industrial applications," *IEEE Trans. Syst., Man and Cybern., Part B: Cybernetics*, vol. 38, no. 3, pp. 743-763, Jun. 2008.