Journal of Global Optimization **27:** 105–129, 2003.

A Trigonometric Mutation Operation to Differential Evolution

HUI-YUAN FAN[1,2] and JOUNI LAMPINEN[2]
[1]*School of Energy and Power Engineering, Xi'an Jiaotong University, Xi'an, 710049, PR China;*
[2]*Department of Information Technology, Lappeenranta University of Technology, P.O. Box 20, FIN-53851 Lappeenranta, Finland E-mail: jlampine@lut.fi*

Abstract. Previous studies have shown that differential evolution is an efficient, effective and robust evolutionary optimization method. However, the convergence rate of differential evolution in optimizing a computationally expensive objective function still does not meet all our requirements, and attempting to speed up DE is considered necessary. In this paper, a new local search operation, trigonometric mutation, is proposed and embedded into the differential evolution algorithm. This modification enables the algorithm to get a better trade-off between the convergence rate and the robustness. Thus it can be possible to increase the convergence velocity of the differential evolution algorithm and thereby obtain an acceptable solution with a lower number of objective function evaluations. Such an improvement can be advantageous in many real-world problems where the evaluation of a candidate solution is a computationally expensive operation and consequently finding the global optimum or a good sub-optimal solution with the original differential evolution algorithm is too time-consuming, or even impossible within the time available. In this article, the mechanism of the trigonometric mutation operation is presented and analyzed. The modified differential evolution algorithm is demonstrated in cases of two well-known test functions, and is further examined with two practical training problems of neural networks. The obtained numerical simulation results are providing empirical evidences on the efficiency and effectiveness of the proposed modified differential evolution algorithm.

Key words: differential evolution, evolutionary algorithm, mutation operation, nonlinear optimization

1. Introduction

Evolutionary algorithms (EAs) are a class of nonlinear optimization approaches based on natural selection and Darwin's main principle: survival of the fittest. EAs have some particular advantages, such as robustness, parallelism, global search capability, etc., which make EAs applicable and very attractive within a wide range of engineering disciplines. The differential evolution algorithm (DE), developed by Storn and Price a few years ago, is one of the most promising new EAs; see Storn and Price [14, 16, 17], Price [8, 9], Lampinen [3]. This method has been demonstrated to be an efficient, effective and robust optimization method that outperforms some of the traditional EAs as reported by Storn and Price in [15]. Furthermore, DE

can be easily extended to handling continuous, discrete and integer variables and for handling of multiple non-linear and non-trivial constraints, see, for example, Lampinen and Zelinka [4]. In addition, DE is extremely compact. Its implementation requires only about 20 lines of code in C or FORTRAN, for example. For these reasons, DE is one of the most attractive evolutionary optimization algorithms for practical engineering optimization work.

Despite DE having a relatively high convergence performance in comparison with the other EAs for nonlinear optimization of multi-modal functions, its convergence velocity is still too low for optimizing the computationally most expensive objective functions. Generally, seeking the global optimum of a multi-modal objective function by any nonlinear optimization algorithm will always call for higher convergence velocity. On the other hand, since all EAs, including DE, work with a population of solutions rather than a single solution, typically a high number of candidate solutions must be evaluated during the optimization process. Unfortunately, for many real-world problems, evaluating a candidate solution is not too difficult, but time-consuming. This will often result in the overall optimization time being too long to be acceptable in finding the global extreme of the objective function despite that the objective function is only moderately multi-modal, providing a relatively smooth function landscape. While DE can solve such problems with high accuracy, the optimization time available does not make it possible to find even a good quality sub-optimal solution. For example, optimization in connection with various finite element methods (FEM), like optimization in the field of computational mechanics, computational magnetics or computational fluid dynamics (CFD), often leads to a computational expensive objective function as described in [10–12]. The objective function evaluation may take several minutes, several hours, even several days. In such cases, there is no other option but to limit the algorithm to operate within an acceptable time, which may not be enough to find the global optima, but enough to obtain an improved solution (i.e., sub-optimum). Thus it is obvious that further study of strategies that can “locally” speed up the current DE algorithm is still an important and interesting topic. Such speed acceleration may allow better solutions to be obtained within the limited CPU-time available for optimization. Furthermore, it may define to what extent the algorithm can be extended into complicated real-world problems with computationally expensive objective functions.

In this article a trigonometric mutation operation is proposed and embedded into the DE algorithm. This modification can provide a measure to tune the balance between the convergence rate and the robustness of the algorithm. As a result, it is expected that the convergence velocity of the DE algorithm can be locally accelerated so that better solutions can be obtained within an acceptable convergence time.

2. A short overview of DE

Currently, there are several variants of the DE-algorithm [9]. The particular version subject to our investigation is the *DE/rand/1/bin*-version, which appears to be the most frequently used variant, and is often considered as the "basic" version of the DE-algorithm. This particular scheme will be briefly described as follows:

When DE is applied to solving an optimization problem formulated as, $\min_X f(X)$, where X is a vector of $n \times 1$ parameters, it works with a population of N_p candidate solutions, i.e., $X_{i,G}$, $i = 1, \ldots, N_p$, where i indexes the population and G is the generation to which the population belongs. The main operations of DE, mutation, crossover and selection, are briefly discussed in turn.

The main difference between DE and other EAs is the implementation of the mutation operation. The mutation operation of DE applies the vector differentials between the existing population members for determining both the degree and direction of perturbation applied to the individual subject of the mutation operation. The mutation process at each generation begins by randomly selecting three individuals in the population. The ith perturbed individual, $V_{i,G+1}$, is therefore generated based on the three chosen individuals as follows:

$$V_{i,G+1} = X_{r_3,G} + F \cdot (X_{r_1,G} - X_{r_2,G}) \tag{1}$$

where, $i = 1, \ldots, N_p$, $r_1, r_2, r_3 \in \{1, \ldots, N_p\}$ are randomly selected and satisfy: $r_1 \neq r_2 \neq r_3 \neq i$, and a scaling factor $F (F \in [0, 1+])$ introduced by Storn and Price [14] in Equation (1) is a control parameter of the DE algorithm.

The perturbed individual, $V_{i,G+1} = (v_{i,j,G+1}, \ldots, v_{n,i,G+1})$, and the current population member, $X_{i,G} = (x_{1,i,G}, \ldots, x_{n,i,G})$, are then subject to the crossover operation, that finally generates the population of candidates, or "trial" vectors, $U_{i,G+1} = (u_{1,i,G+1}, \ldots, u_{n,i,G+1})$, as follows:

$$u_{j,i,G+1} = \begin{cases} v_{j,i,G+1} & \text{if } rand_j \leqslant C_r \vee j = k \\ x_{j,i,G} & \text{otherwise} \end{cases} \tag{2}$$

where, $i = 1, \ldots, N_p$, $j = 1, \ldots, n$, $k \in \{1, \ldots, n\}$ is a random parameters index, chosen once for each i, and the crossover factor, $C_r \in [0, 1]$, the other control parameter of DE, is set by the user.

The selection scheme of DE also differs from that of other EAs. The population for the next generation is selected from the individual in current population and its corresponding trial vector according to the following rule:

$$X_{i,G+1} = \begin{cases} U_{i,G+1} & \text{if } f(U_{i,G+1}) \leqslant f(X_{i,G}) \\ X_{i,G} & \text{otherwise} \end{cases} \tag{3}$$

Thus, each individual of the temporary (trial) population is compared with its counterpart in the current population. The one with the lower objective function

value will survive from the tournament selection to the population of the next generation. As a result, all the individuals of the next generation are as good or better than their counterparts in the current generation. An interesting point concerning DE's selection scheme is that a trial vector is not compared against all the individuals in the current generation, but only against one individual, its counterpart, in the current generation.

3. Trigonometric Mutation Operation

A mutation operation of DE is performed usually on the basis of three individuals in the current population. An individual, being taken as a donor, is perturbed with a scaled vector differential from the other two individuals so as to produce a mutated individual. This enables a hypergeometric triangle to be formed in the search space where the three selected individuals (i.e., points) exist as the vertices, and to use the selected-individuals' objective function values together with this triangle in a mutation operation. By applying also the objective function value information in the mutation operation the perturbations can be biased towards the points (the vertices of the hypergeometric triangle) providing the lowest objective function values.

From this, a new mutation approach, namely, a trigonometric mutation operation, is proposed and described as follows. Assume the three mutually different individuals, $X_{r_1,G}$, $X_{r_2,G}$ and $X_{r_3,G}$, are randomly chosen to implement a mutation operation for the i-th individual $X_{i,G}$, where $r_1, r_2, r_3 = \{1, \ldots, N_P\}$ are randomly selected and satisfy: $r_1 \neq r_2 \neq r_3 \neq i$. When a trigonometric mutation operation is performed, instead of an individual randomly taken from the three chosen ones as in the original mutation of DE, the donor to be perturbed is taken to be the center point of the hypergeometric triangle. The perturbation to be imposed to the donor is then made up with a sum of three weighted vector differentials. The mutation operation is performed according to the following formulation:

$$\begin{aligned} V_{i,G+1} = {} & (X_{r_1,G} + X_{r_2,G} + X_{r_3,G})/3 + (p_2 - p_1)(X_{r_1,G} - X_{r_2,G}) \\ & + (p_3 - p_2)(X_{r_2,G} - X_{r_3,G}) + (p_1 - p_3)(X_{r_3,G} - X_{r_1,G}) \end{aligned} \tag{4}$$

where:

$$\begin{aligned} p_1 &= |f(X_{r_1,G})|/p' \\ p_2 &= |f(X_{r_2,G})|/p' \\ p_3 &= |f(X_{r_3,G})|/p', \end{aligned}$$

and

$$p' = |f(X_{r_1,G})| + |f(X_{r_2,G})| + |f(X_{r_3,G}))|$$

As it can be seen from the above formulation, the perturbation part in the trigonometric mutation is contributed together by the three legs of the triangle defined

with $X_{r_i,G}$, $i = 1, 2, 3$, i.e., by $(X_{r_1,G} - X_{r_2,G})$, $(X_{r_2,G} - X_{r_3,G})$ and $(X_{r_3,G} - X_{r_1,G})$. This perturbation can also equivalently be viewed as yielded by the donor, namely, the triangle's center, through shifting along the directions of each leg of the triangle with different step-sizes respectively. The weight terms $(p_2 - p_1)$, $(p_3 - p_2)$ and $(p_1 - p_3)$ to the vector differentials are defined along the following two considerations. Firstly, weight terms can make the perturbation have a tendency to produce a better individual. This can further be explained from the case that the perturbation is viewed as the result from the donor's shifts. Evidently, with these weight terms the donor is insured to move along in the direction from a vertex with a higher value of objective function towards a vertex with a lower value of objective function. Secondly, the weight terms can automatically scale the contribution magnitudes of the vector differentials to the perturbation in such a way that the greater the difference in the objective function values between the individuals that form a vector differential, the larger the contribution the corresponding vector differential offers to the perturbation.

Through the above analysis it can easily be observed that the trigonometric mutation operation is a rather greedy operator since it biases the new trial solution strongly in the direction where the best one of three individuals chosen for the mutation is. Therefore the trigonometric mutation can be viewed as a local search operator.

From the definition of trigonometric mutation it can be deduced that the perturbed individuals are produced only within a trigonometric region that is defined by the triangle used for a mutation operation. A two-dimensional minimization problem is taken as an example to illustrate this fact. As shown in Figure 1, if the three individuals, $X_{r_1,G}$, $X_{r_2,G}$, and $X_{r_3,G}$ (here $X_{r_1,G} = (2, 2)$, $X_{r_2,G} = (6, 4)$, and $X_{r_3,G} = (3, 6)$, for instance), are chosen to produce a perturbed individual, the trigonometric region is determined by the following procedure. Considering an extreme case of $p_1 = 1$, or equivalently, $p_2 + p_3 = 0$, from Equation (4), the point, X'_{r_1}, can be determined as

$$X'_{t_1} = (X_{r_1,G} + X_{r_2,G} + X_{r_3,G})/3 + X_{r_2,G} + X_{r_3,G} - 2X_{r_1,G}).$$

Similarly, the other two points X'_{r_2} and X'_{r_3} can be determined with respects to the other two extreme cases of $p_2 = 1$ and $p_3 = 1$ respectively as

$$X'_{r_2} = (X_{r_1,G} + X_{r_2,G} + X_{r_3,G})/3 + X_{r_1,G} + X_{r_3,G} - 2X_{r_2,G})$$

and

$$X'_{r_3} = (X_{r_1,G} + X_{r_2,G} + X_{r_3,G})/3 + X_{r_1,G} + X_{r_2,G} - 2X_{r_3,G}).$$

It can be proven that the three points, X'_{r_1}, X'_{r_2}, and X'_{r_3} just form the trigonometric region that limits mutation results. Any resultant individual by the trigonometric mutation based on the three chosen individuals will not be outside this region.

The basic idea of the trigonometric mutation operation can be further clarified by Figure 1. Here it is assumed $p_1 < p_2$, $p_1 < p_3$, $p_2 < p_3$. A mutation operation

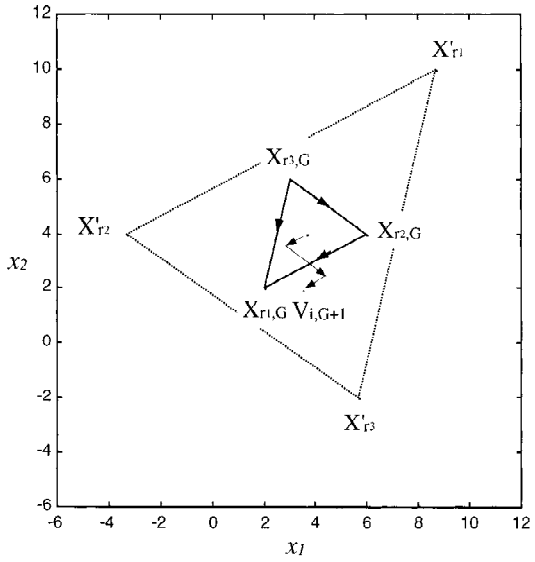

Figure 1. The trigonometric mutation operation illustrated for the case of a two-dimensional function minimisation problem: $X_{r_i,G}$, $i = 1, 2, 3$, are the individuals chosen to mutate the i-th individual of the current population; $V_{i,G+1}$ is a possible perturbed individual by trigonometric operation; an arrowhead on the legs of the triangle defined by $X_{t_1,G}$, $i = 1, 2, 3$, indicates the direction of decreasing objective function values between two chosen individuals; $X'_{r_i,G}$, $i = 1, 2, 3$, indicates the direction of decreasing objective function values between two chosen individuals; $X'_{r_i,G}$, $i = 1, 2, 3$, which is defined by $X_{r_i,G}$, $i = 1, 2, 3$, which forms a triangular region that limits the trigonometric mutation to produce a perturbed individual only within that region.

starts from the canter (i.e., donor) of the triangle used for the mutation operation, and makes the donor shift along in the direction from $X_{r_2,G}$ to $X_{r_1,G}$, from $X_{r_3,G}$ to $X_{r_2,G}$, and from $X_{r_3,G}$ to $X_{r_1,G}$, respectively, with step-sizes depending on the respective objective function values, and finally yields a perturbed individual. Evidently, this process makes an individual mutated to an area where the individuals are likely to have lower objective function values.

In order to make a further observation to the trigonometric mutation operation, three representative cases of the different p_i, $i = 1, 2, 3$, i.e., Case One: $p_1 < p_2$, $p_1 < p_3$, $p_2 < p_3$; Case Two: $p_1 < p_2$, $p_1 < p_3$, $p_2 > p_3$; and Case Three: $p_1 < p_2$, $p_1 < p_3$, $p_2 = p_3$, are studied by randomly producing 200 sets of different values of p_i, $i = 1, 2, 3$, for each case. The perturbed individuals obtained through the trigonometric mutation operation are plotted in Figure 2. It can be observed that the trigonometric mutation operation tends to bias the mutated individuals towards an optimal region that is locally and roughly determined on the basis of the three chosen individuals and their objective function values.

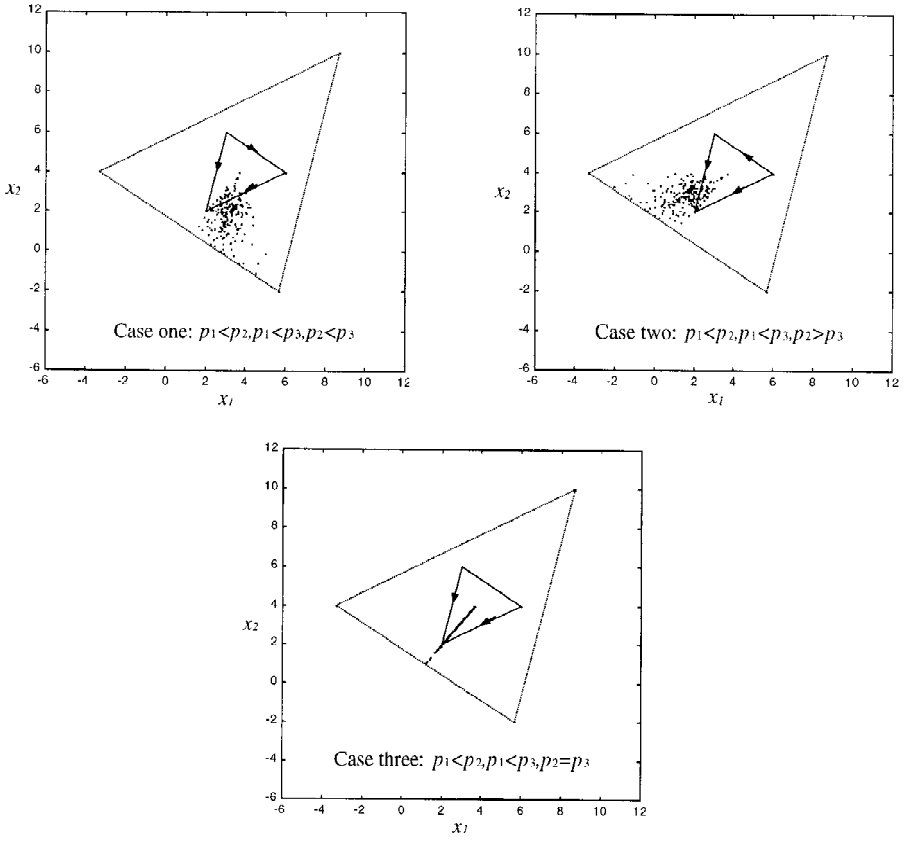

Figure 2. The trigonometric mutation operation illustrated in three representative cases for a two-dimensional minimisation problem. Two hundred perturbed individuals are plotted. These points are obtained by randomly generating the values for p_i, $i = 1, 2, 3$. The values are required to satisfy the above-mentioned conditions for each case.

Since all of the variables in Equation (4) are known at the moment of applying the trigonometric mutation, determining the new trial individual with Equation (4) is a straightforward procedure.

4. The modified DE algorithm

In this paper the trigonometric mutation operation is embedded into the original *DE/rand/1/bin*-version to form a new modified DE algorithm. Later on the modified DE algorithm will be referred to as the TDE (trigonometric mutation differential evolution) algorithm. The TDE algorithm can be outlined as follows:

The TDE Algorithm:

1. Initialization of the population.

2. Mutation operation:
   2.1 Perform trigonometric mutation with Equation (4) with a probability M_t, or
   2.2 perform original mutation with Equation (1) (with a probability $1 - M_t$).
3. Crossover operation.
4. Evaluation of the population with the objective function.
5. Selection.
6. Repeat from step 2 to 5.

Note here, that the only structural difference between the TDE algorithm and the original DE algorithm is the new trigonometric mutation operation added into the TDE. This extra step is controlled by a new control parameter "trigonometric mutation probability", M_t. Thus, M_t is an extra search control parameter of TDE. As it is well known, hybridizing a local search operator into an evolutionary algorithm can generally speed up the convergence velocity of the algorithm. However, such an operator usually leads to a greedy algorithm prone to converge prematurely into a local optimum. Therefore, the frequency of performing a local search operation in an EA must be appropriately controlled in order to allow the algorithm to keep a good balance between fast convergence and global optimum searching ability. In the TDE algorithm, the parameter M_t provides a convenient way to adjust this balance. It is also useful to note here a special case, $M_t = 0$, when TDE effectively reduces into original DE.

The trigonometric mutation operation is used jointly with the DE's original mutation operation so that only one or the other is applied for generating the current trial individual, depending on a uniformly distributed random value within the range [0.0, 1.0]. If the random value is smaller than M_t, then a trigonometric mutation is applied using Equation (4), otherwise the mutation is performed according to Equation (1).

5. Illustrative examples

5.1. TEST FUNCTIONS AND ARRANGEMENTS

Two test functions that are known to be densely multi-modal were chosen to demonstrate the TDE algorithm. The first is so called Ackley's function, as it can be found in [1, 2]. This is a continuous, highly multi-modal function that causes the search with moderate complications. The second is commonly called as Rastrigin's function, and can be found in [7]. This function is also considered relatively difficult to minimize because the number of locally optimum points is high. These two test functions are defined as follows:

Ackley's function [1, 2]:

$$f_1(X) = -20 \cdot \exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i\right) + 20 + e;$$
$$-20 \leqslant x_i \leqslant 30, n = 30 \tag{5}$$

The global minimum is $f_1(X^*) = 0$ with $x_i = 0, i = 1, 2, \ldots, n$.

Rastrigin's function [7]:

$$f_2(X) = 2n + \sum_{i=1}^{n}(x_i^2 - 2\cos(2\pi x_i));$$
$$-5.12 \leqslant x_i \leqslant 5.12, n = 20 \tag{6}$$

The global minimum is $f_2(X^*) = 0$ with $x_i = 0, i = 1, 2, \ldots, n$.

The performance of the DE algorithm, of course, depends on how the values for its control parameters, namely, F, C_r, N_p, are chosen. As it is well known, the aforementioned parameters are problem-specific. On how to choose their values optimally *a priori* is still a partially open problem. Usually, a trial-and-error approach can be used to complete a sub-optimal selection of these parameters, but it is rather laborious and time-consuming. In consideration of a good generality, in this investigation a basic set of these control parameters was first chosen based on the authors' experience and the general recommendations given in the literature, for example, in [3, 9, 13, 14, 16–18]. Then, all the numerical simulations and result comparisons were carried out around this basic parameter set, namely: $F = 0.99$, $C_r = 0.85$, $N_p = 30$. The special control parameter of TDE, i.e., the trigonometric mutation probability , was chosen as $M_t = 0.05$ after performing a set of experiments that will be discussed in detail in the following section.

In general, in the numerical experiments, the population of a DE algorithm was always randomly initialized. Despite this, both the TDE algorithm and the original DE algorithm to be compared were tested by applying identical initial populations. In other words, when an initial population was once randomly generated, it was then used as a starting point for a single trial run both for the TDE and the DE. Thus, both the compared algorithms were tested with identical, but still randomly generated initial populations.

Due to the stochastic nature of DE algorithm, for both of the compared algorithms, the experiments were repeated multiple times for each test function (10–50 times depending on the case). When applicable, the results subject to our interest were averaged over all the repeated experiments. A case dependently specified maximum CPU-time was applied as stopping criteria. For the first test function, the given maximum CPU-time was 45 s, and for the second, 30 s.

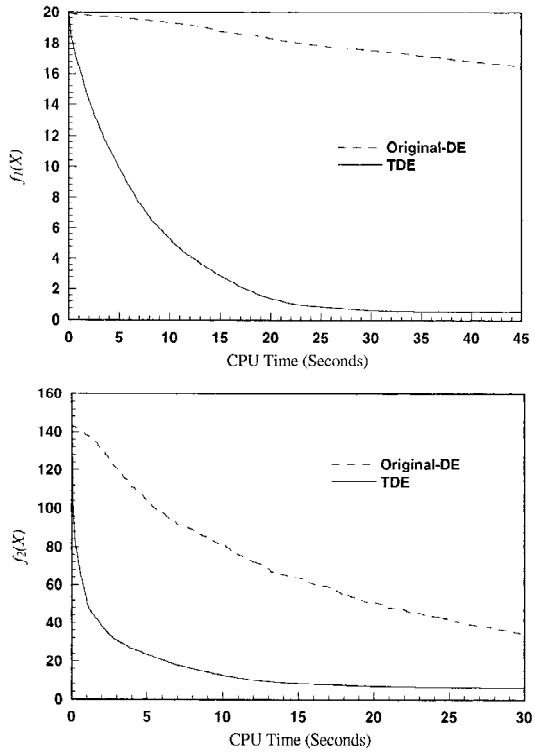

Figure 3. Convergence histories of the TDE algorithm and the original DE algorithm for minimization of the test functions f_1 and f_2. The results are averaged over 50 independent experiments for each algorithm.

Both the TDE algorithm and the original DE algorithm were implemented within Matlab 5.0 system and were executed in PC with a Pentium III processor.

5.2. OVERALL RESULTS

Some overall performance measures of the compared algorithms were examined first. Figure 3 shows the convergence histories of the TDE algorithm and the original DE algorithm for minimization of the two test functions. The results were averaged over 50 independent trial runs for each algorithm for each test function. From Figure 3 it can be observed that for both functions the TDE expressed a considerably higher convergence velocity than the DE did. Though the TDE finally stagnated into a sub-optimal solution respectively as also did the original DE algorithm, the TDE converged faster in the earlier generations. Nevertheless, the sub-optimal solutions obtained by the TDE were closer to their corresponding global minimum than those obtained by the DE.

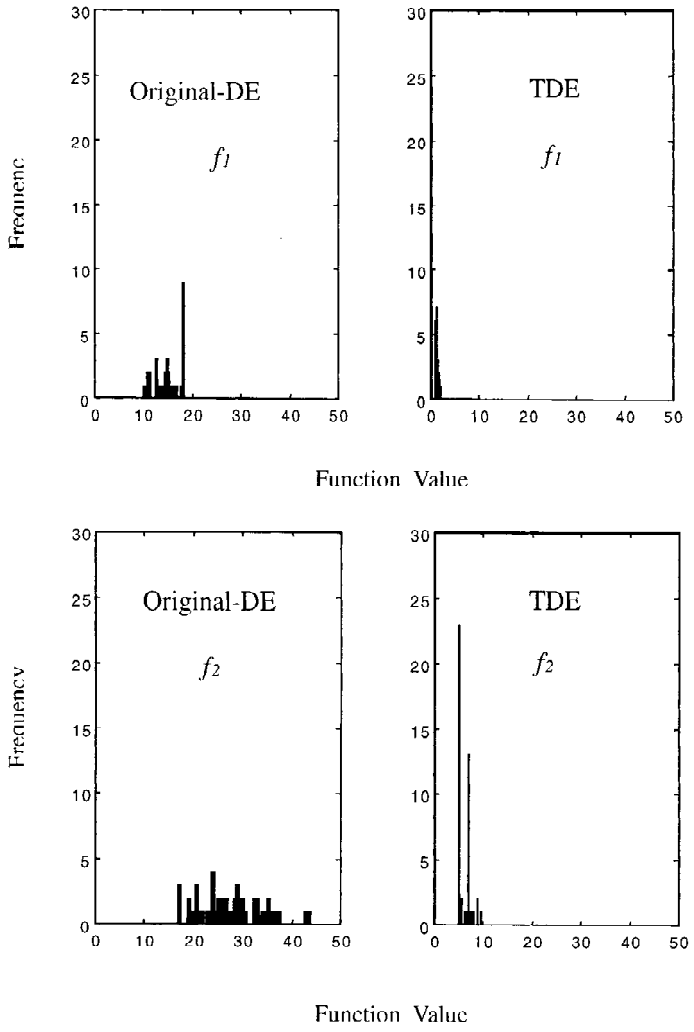

Figure 4. Histograms of the function values obtained by the TDE algorithm and the original DE algorithm in the case of 50 independent trial runs.

The convergence reliability of the TDE algorithm was further assessed by histograms (Figure 4) representing the obtained function values from 50 independent trial runs for both the test functions, and for both the TDE and the DE algorithms. From Figure 4 it can be observed that for both chosen test functions the TDE continuously converged (in all experiments) into a closer vicinity of the true global optima than the best run of the DE. Within the given maximum CPU-time the DE was able to find only sub-optimal solutions that were still quite far from the real minimum function values. The results in Figure 4 qualitatively suggest that the

TDE algorithm has a better convergence reliability than the original DE algorithm in minimization of the applied test functions.

5.3. TUNING THE CONTROL PARAMETER, m_t

The TDE algorithm has the extra control parameter for the trigonometric mutation probability M_t. The influence of M_t to the convergence performance of the TDE was also investigated with a set of numerical experiments. For this purpose, the basic values of the control parameters F, C_r and N_p as described in Section 5.1 were applied, and then the TDE was applied to minimize the two test functions by experimenting with 17 different values for M_t. For each value of M_t, 50 independent trial runs were performed with the TDE. The function values obtained by the TDE were recorded and averaged over these 50 trial runs. From the Figure 5 it can be found that that there appears to be an optimal value for both of the applied test functions: for Ackley's function, this value is about 0.03, and for Rastrigin's function it is about 0.05. Since these two optimal values are rather close to each other, for simplification, we considered $M_t = 0.05$ as an appropriate value for both the test functions, and used this value also for all our further experimentation to be discussed later on.

The convergence histories of the TDE in the case for minimizing the Ackley's function, corresponding to eight out of all 17 experimented values of M_t, are shown in Figure 6 to demonstrate the influence of M_t to the convergence performance of the TDE. An important detail that can be observed from the above results is an early stagnation tendency of the TDE algorithm while increasing the value of M_t. This supports further on the initial assumption that the trigonometric mutation proposed here is effectively a local search operator. The larger the M_t parameter, the greedier is the algorithm, and the higher is the probability of premature convergence into a local optima. For example, when M_t is close to one, the algorithm quickly stagnates far from the global optimum. However, also the convergence rate appears to increase along with the value of M_t. Indeed, the TDE appear to offer a useful mean for accelerating convergence through tuning of the added parameter, M_t, and thereby adjust the desired balance between the convergence rate and failure probability. Note, that generally a high convergence rate and low failure probability are conflicting objectives. Nevertheless, some care must be taken to select the value of M_t properly. The results here suggest that the value of M_t should be rather low, $M_t = 0.05$ appears to be a good initial setting to be tried first.

5.4. SENSITIVITIES TO OTHER CONTROL PARAMETERS AND TO PROBLEM SIZE

The influence of the other control parameters on the TDE algorithm was also investigated and compared with the corresponding results of the original DE al-

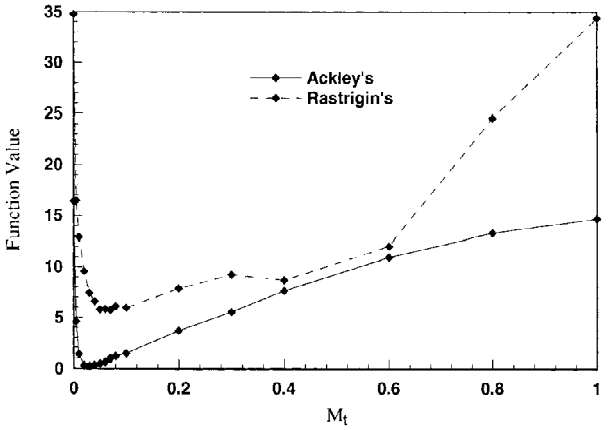

Figure 5. With different values of trigonometric mutation probability M_t, the TDE algorithm obtained different test function values within the specified maximum CPU-time.

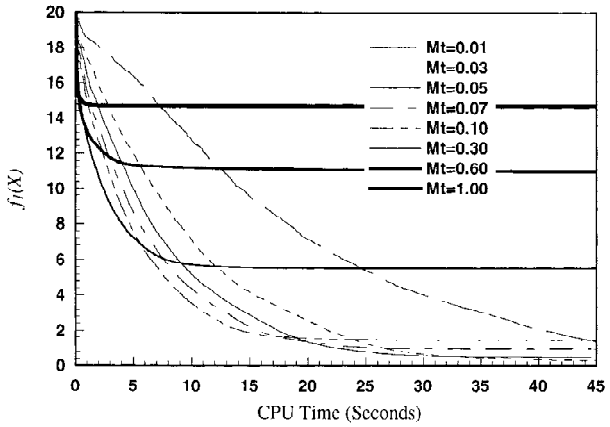

Figure 6. Convergence histories of the TDE algorithm with different values of trigonometric mutation probability M_t in the case for minimizing the Ackley's function.

gorithm. During the experimentation, one among the three parameters, F, C_r, N_p, was varied stepwisely within a certain range, while the remaining ones were kept in their basic values, as described in Section 5.1. For each individual set of control parameter values 20 independent trial runs were performed with both the TDE and DE. The obtained function values were recorded and averaged over all the trial runs performed. The results of these experiments are shown in Figures 7–9.

From Figures 7–9 it can be seen that the TDE indicated a considerably lower sensitivity to the variations in search control parameters, than the original DE did. For all control parameters, F, C_r and N_p, the DE appears to require selecting a value from a noticeably narrower range than the TDE does. Due to it's indicated lower sensitivity to the control parameter values, from a user point of view, the usage of the TDE can be expected to be simpler than the usage of the DE. The

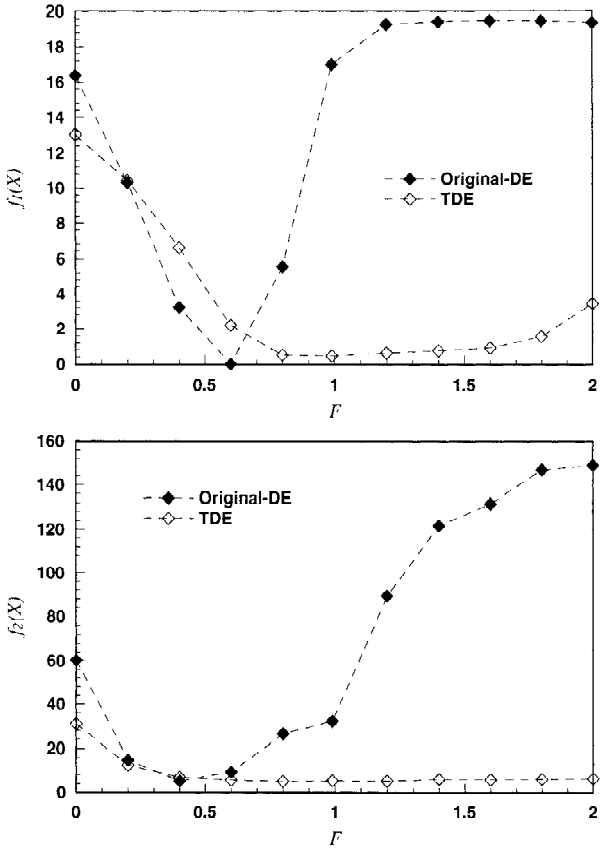

Figure 7. The function values obtained by the TDE algorithm and the original DE algorithm with different values of scaling factor F in minimization of the test functions. The TDE demonstrated a lower sensitivity to the value of F in both cases. The results are averages over 20 independent trial runs for each point plotted.

control parameter values are easier to set by the user *a priori*. In brief, the TDE algorithm allows these three parameters, F, C_r and N_p, to be selected from a larger range than the original DE algorithm does, without major performance degradation. While there is an extra search control parameter, M_t, in the TDE, the rest of the search parameters, F, C_r and N_p, are easier to set due to their lower sensitivity in comparison with the original DE.

The influence of the problem size (dimensionality) to the performance of the TDE and DE algorithms was also initially and preliminarily studied. In order to investigate this, the basic control parameter settings as described in Section 5.1 were applied, and used for both algorithms to minimize the two test functions varying the function dimensionality (both functions have scalable dimensionality). With each given number of parameters, both algorithms were applied repeatedly and independently twenty times. The function values obtained by the TDE and the

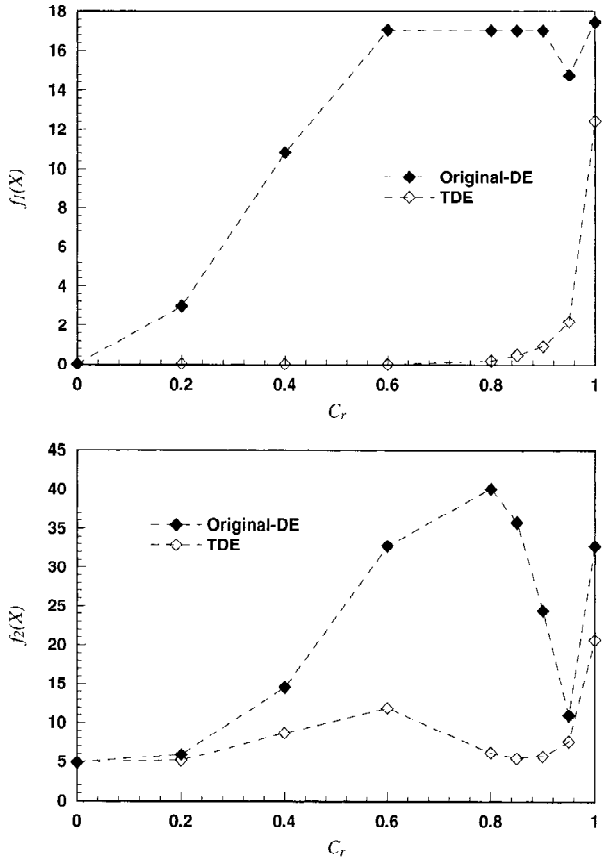

Figure 8. The function values attained by the TDE algorithm and the original DE algorithm for different values of crossover factor C_r. The TDE demonstrated a lower sensitivity to the value of C_r in both cases. The results are averages over 20 independent trial runs for each point plotted.

DE were recorded and averaged over all the performed trial runs. From the results shown in Figure 10 it can be observed that when the test function's dimensionality was increased, the function values obtained by the TDE algorithm still remained clearly below those obtained by the original DE. These results did not revealed any major problems with the scalability of the DE or TDE to higher dimensional problems.

Please notify here, that our current results described in this section are mainly based on the results obtained from numerical simulations. Due to the contemporary shortage of the mathematical analysis of the dynamic behavior of the DE algorithm, it was not possible to clarify and to theoretically support all observations based on the numerical results as adequately as desired. For example, why the TDE algorithm indicated a much lower sensitivity to the control parameters than the original DE algorithm? The phenomena may depend on the algorithm's dynamic

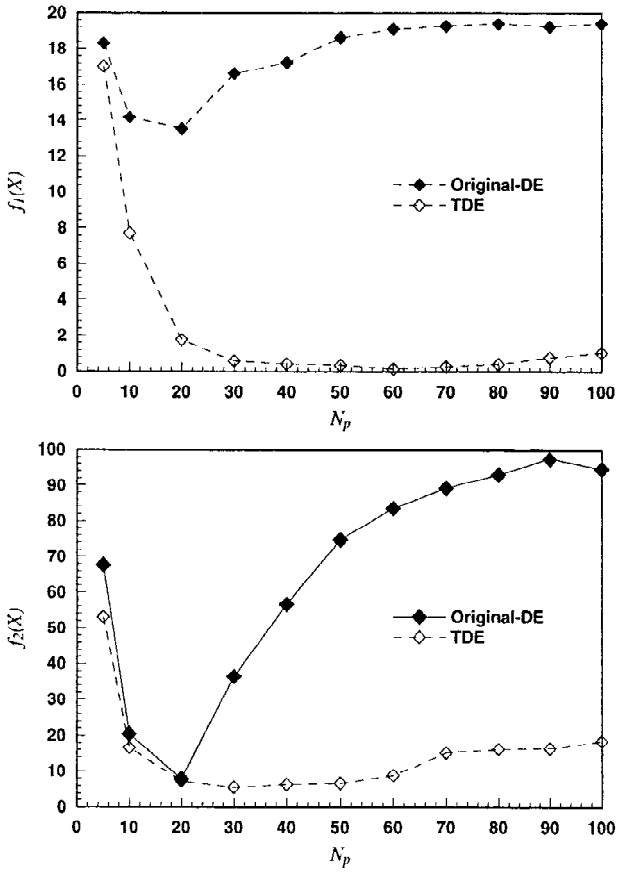

Figure 9. The function values attained by the TDE algorithm and the original DE algorithm for different values of crossover factor N_p. The TDE demonstrated a lower sensitivity to the value of N_p in both cases. The results are averages over 20 independent trial runs for each point plotted.

characters or on the problems themselves. All these questions still remain open for further studies in the future.

6. Application for training neural networks

The neural network technique is a computation strategy that simulates the biological process in the human brain. A neural network consists of simple, highly interconnected processing elements called neurons. Neurons by themselves are not particularly interesting, but their interconnection creates a powerful device that is proven to be capable of approximating arbitrary functions. Neural networks have been shown to successfully model a wide diversity of practical systems and processes.

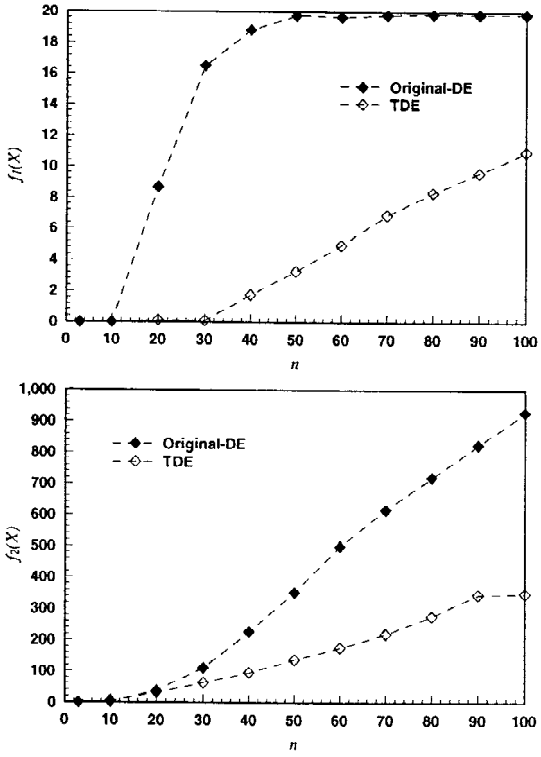

Figure 10. The function values obtained by the TDE algorithm and the original DE algorithm as a function of the problem size, n. The Ackley's function, f_1, and the Rastrigin's function, f_2, were used as test functions to be scaled for different problem dimensionalities. Note, that both test functions can be easily scaled for any number of dimensions.

However, the effective training of a neural network that is used to approximate a complicated practical system can still be viewed as one of the most important open problems in the field of neural networks. A training problem of a neural network usually can be attributed to a multi-modal nonlinear optimization problem, often having hundreds of parameters. A wide variety of difficulties have been encountered in solving such a problems. A low convergence speed allowing only a minor convergence within the time available for computation, for example, is one of the most common practical difficulties. This is because a practical physical system is usually so complex that its approximation by a neural network may need a large network structure as well as a large number of samples to train it. In such a case the very slow speed of convergence for the training process often becomes a fatal drawback for the neural network approach. As a result, any attempts made to improve the convergence of neural network training should be very profitable to real neural network problems.

In this section the proposed TDE algorithm is applied to the training problems of multi-layered feed forward neural networks to further demonstrate its convergence speeding-up ability.

6.1. SOME PROBLEM-SPECIFIC ADAPTATIONS

Let Y and $O = g(Y)$ be the input and output vectors for the system g to be approximated by a multi-layered feed forward neural network from a set of known input-output pairs, i.e., samples, $(Y, O)_j$, $j = 1, \ldots, N$. During the training, the neural network "black-box" is presented with the training pairs $(Y, O)_j$ and its internal parameters (connecting weights) are adjusted by the training algorithm. This procedure results in the "encoding" of the properties of the function g in different parts of the neural network. After training is completed, the neural network becomes an approximator of the system. When the trained neural network is presented with an input, Y, it will simulate the function g and will produce an output O' that is an approximation of the target output, O. Without loss of generality, it is assumed that a feed forward neural network with $(M+1)$ layers will be trained. Denote the connecting weight matrix between the layers of the i-th to the $(i+1)$-th as $W^{[i]}$, $i = 1, \ldots, M$, and further, let $W = \{W^{[1]}, \ldots, W^{[M]}\}$. The training of the neural network can be interpreted as a minimization process of a proper error function of the neural network. For neural network training, the commonly used output mean squared error (MSE) of the neural network was applied here as an error function. The training process can then be formulated as the following minimization problem:

$$\min_{W} \left\{ \varphi(W) = \frac{1}{N} \sum_{j=1}^{N} \left[\sum_{i=1}^{m} (O_{ji} - O'_{ji})^2 \right] \right\} \tag{7}$$

where m is the dimension of the output vectors, or equally, the number of output neurons.

When DE is applied to solve the above minimization problem, the individuals are no longer vectors, but groups of weight matrices of the neural network. Of course, the weight matrixes in a group can be reshaped to vectors and thereby be formed into a single vector through linking the reshaped vectors end by end in a proper sequence so as that the previously prescribed DE operations can be directly used without any adaptations. However, some additional work may be added into the training process. For example, inverse reshape work is needed in order to evaluate an individual's objective function value. In view of this fact, the presentations of individuals with matrixes are reserved into the DE's operations in this application. Due to this, the DE algorithm must evolve in synchronism a population in which the individuals, i.e., $W_{i,G}$, $i = 1, \ldots, N_p$, are groups of weight matrixes with several different sizes. So all the DE operations are required to adapt to operate based on such individuals.

In this section, in correspondence with the fact that the to-be-optimized solutions, or individuals, in the population have already been represented with the sets, $W_{i,G}, i = 1, \ldots, N_p$, the perturbed individuals and the trial individuals are also newly denoted with $R_{i,G}, i = 1, \ldots, N_p$, and $S_{i,G}, i = 1, \ldots, N_p$, respectively. The three DE operations, as adapted to the training problem of the neural network, i.e., the minimization problem as Equation (7), will be as follows:

The mutation operation can expressed as $R_{i,G+1} = \{R^{[1]}_{i,G+1}, \ldots, R^{[M]}_{i,G+1}$, in which, $R^{[1]}_{i,G+1}, l = 1, \ldots, M$, are defined respectively for the original mutation and the trigonometric mutation as follows:
For the original mutation:

$$R^{[l]}_{i,G+1} = W^{[l]}_{r_3,G} + F \cdot (W^{[l]}_{r_1,G} - W^{[l]}_{r_2,G}) \tag{8}$$

For the trigonometric mutation:

$$\begin{aligned} R^{[l]}_{i,G+1} =& (W^{[l]}_{r_1,G} + W^{[l]}_{r_2,G} + W^{[l]}_{r,G})/3 + (p_2 - p_1)(W^{[l]}_{r_1,G} - W^{[l]}_{r_2,G}) \\ &+ (p_3 - p_2)(W^{[l]}_{r_2,G} - W^{[l]}_{r_3,G} + (p_1 - p_3)(W^{[l]}_{r_3,G} - W^{[l]}_{r_1,G}) \end{aligned} \tag{9}$$

where, $p_1 = \varphi(W_{r_1,G})/p'$, $p_2 = \varphi(W_{r_2,G})/p'$, $p_3 = \varphi(W_{r_3,G})/p'$, and $p' = \varphi(W_{r_1,G}) + \varphi(W_{r_2,G}) + \varphi(W_{r_3,G})$.

Similar to the mutation operation, the crossover operation can also be expressed as $S_{i,G+1} = \{S^{[1]}_{i,G+1}, \ldots, S^{[M]}_{i,G+1}\}$. Let $w^{[}l]_{i,G}(m,n)$, $r^{[l]}_{i,G}(m,n)$, represent the elements in the m-th row and the n-th column of $W^{[l]}_{i,G}$, $R^{[l]}_{i,G}$, $S^{[l]}_{i,G}$, respectively, then $S^{[l]}_{i,G+1}, l = 1, \ldots, M$, in the equation are defined as:

$$s^{[l]}_{i,G+1}(m,n) = \begin{cases} r^{[l]}_{i,G+1}(m,n) & \text{if } rand_{m,n}[0.1] \leqslant C_r \vee (m,n) = (m^*, n^*) \\ w^{[l]}_{i,G+1}(m,n) & \text{otherwise} \end{cases} \tag{10}$$

where $m = 11, \ldots, M_l$, $n = 1, \ldots, N_l$, M_l and are the row size and the column size of the related matrix $[\bullet]^{[l]}$ (e.g., $W^{[l]}_{i,G}$, $R^{[l]}_{i,G}$, $S^{[l]}_{i,G}$, etc), $m^* \leqslant M_l$, $n^* \leqslant N_l$, are random element indexes, chosen once for each element.

Compared with the above two operations the selection operation is simpler and more similar to the original selection operation as described in Equation (3). It is expressed as:

$$W_{i,G+1} = \begin{cases} S_{i,G+1} & \text{if } \varphi(S_{i,G+1}) \leqslant \varphi(W_{i,G}) \\ W_{i,G} & \text{otherwise} \end{cases} \tag{11}$$

It is also worth mentioning that the problems usually handled by DE are often with the parameters subject to lower and upper boundary constraints, so that the population can easily be initialized with random values chosen within the given boundary constraints. However, when DE is applied to train a neural network, values of the

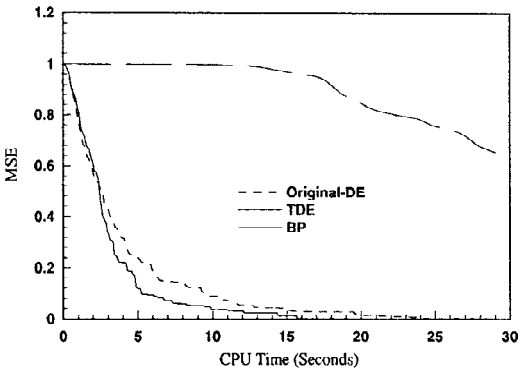

Figure 11. Convergence histories of the TDE algorithm, the original DE algorithm and the standard BP algorithm in training the neural network for the XOR problem. The results are averaged over 50 independent experiments for each algorithm.

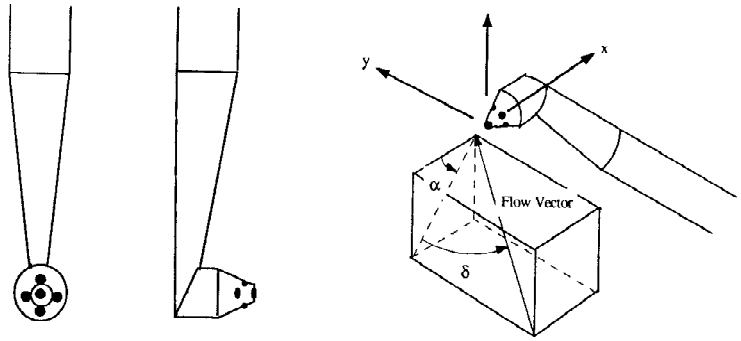

Figure 12. Configuration of the five-hole probe and its coordinate system.

elements for a weight matrix generally need not be constrained by lower and upper boundaries. In order to straightforwardly initialize the population in this case, the uniformly distributed random values in the range $[-1, 1]$ are assigned to all the elements of the weight matrixes.

Having the above adaptations, the existing DE algorithm can be mechanically applied to training neural networks without any difficulties. Except the variables that represent the individuals such as $X_{i,G}$, $V_{i,G}$, and $U_{i,G}$ should be replaced with the variables newly defined as $W_{i,G}$, $R_{i,G}$, and $S_{i,G}$, and all the other evolution parameters remain as the same definitions as in the previous Sections 2–4. The algorithm scheme of DE keeps in the original structure at all times.

6.2. CASE 1: THE XOR PROBLEM

The simple XOR problem was first considered to examine the behaviors of the TDE algorithm through training a neural network to approximate the problem. A four-layered feed forward neural network with the structure formed as "2-2-2-1",

i.e., two input neurons used to input the XOR variables, and the one output neuron used to output the XOR function value, and each of the two hidden layers with two neurons, were taken in this case. The evolution parameters were chosen by trial-and-error as: $N_p = 40$, $C_r = 0.85$, $F = 0.99$. The trigonometric mutation probability was chosen as $M_t = 0.05$ on the basis of the experimental results provided in Section 5.3.

The neural network was trained with four input-output pairs: $([1\ 1]^T, [1])$, $([0\ 0]^T, [1])$, $([1\ 0]^T, [0])$, $([0\ 1]^T, [0])$. Both the TDE algorithm and the original DE algorithm were used to train the network. For a further comparison, the standard Back-Propagation algorithm (BP), implemented as described in [6], was also used to train the network. The maximum CPU-time available for the training was specified to 30s. The resulting convergence histories for the three algorithms are shown in Figure 11. The results are averaged over 50 independent trial runs.

6.3. CASE 2: AN AERODYNAMIC FIVE-HOLE PROBE CALIBRATION

Five-hole probes have established themselves as some of the easiest-to-use and cost-effective devices for three-component velocity measurements of fluids in research as well as in industry environments. By measuring the five pressures at the ports of a probe's tip and through the use of proper probe calibration methods, the five-hole probes can measure the three velocity components, the total and static pressure of a flowfield at the location of its tip. Their calibrations are just the job to establish the mapping relations between the pressure values measured in the five ports and the measured flowfield properties. Generally, these mapping relations are nonlinear. So the neural network is a good option to approximate these relations.

A five-hole probe with a conical tip was chosen in this application case. The configuration of the probe and its coordinate system is shown in Figure 12. In the tip, one port sits at the center of the cone, and the other ports are axisymmetrically arranged in a ring downstream. Generally, a local velocity vector in the measured flowfield can be fully determined with four local flow parameters. They are the flow pitch angle, α, the flow yaw angle, β, the total pressure coefficient A_t, and the static pressure coefficient A_s. Consequently, to calibrate the probe is to determine the above four variables (output variables) as functions of the five measured pressures or equivalently, four non-dimensional pressure coefficients, $B_i, i = 1, 2, 3, 4$, (input variables). Let P_5 denote the pressure measured at the central port, and P_i, $i = 1, 2, 3, 4$, denote the pressures at the other four ports. The pressure coefficients are defined as: $B_i = (P_5 - P_i)/P'$, where $P' = P_5 - 0.25 \cdot \sum_{i=1}^{4} P_i$. When a multi-layered feed forward neural network is applied to fulfill the above calibration task, it can be interpreted as to train a neural network to approximate the mapping relation as

$$A = g(B) \tag{12}$$

where $A = [\alpha \beta A_t A_s]^T$, and $B = [B_1 B_2 B_3 B_4]^T$.

The probe calibration data was obtained on an air jet calibration facility. Varying the α and β in steps of 5° from $-40°$ to $+40°$, respectively, the total number of $17 \times 17 = 289$ data points were obtained that form 289 input-output pairs, i.e., $(B, A)_i$, $i = 1, \ldots 289$. From these input-output pairs 153 were uniformly chosen for the following neural network training.

Two feed forward neural networks, each with four layers, were trained in this case. They had different numbers of hidden neurons, and had structures in the form of "4-10-10-4" and "4-20-20-4", respectively. Here the number of hidden layers and the number of neurons in the layers were selected *a priori* rather arbitrarily based on the author's experience and no attempts were made towards optimizing these selections. Since the aim was to prove the algorithm's local convergence performance and not to find an optimal structure for the neural networks, the two structures were chosen here only for the purpose of an extended observation. Clearly, for both neural networks, the four input neurons represented the four input variables B_i, $i = 1, 2, 3, 4$, while the four output neurons represented the parameters α, β, A_t and A_s.

The training problem of the neural network was solved using the TDE algorithm and the original DE algorithm with control settings of: $N_p = 20$, $F = 0.99$, $C_r = 0.85$. For the TDE, the trigonometric mutation probability M_t was again chosen as, $M_t = 0.05$. For further comparisons, also in this case, the standard BP algorithm [6] was again used to train the neural networks. For the first network, the maximum CPU-time for training was limited to 20 min, while for the second network configuration it was limited to 60 min. The convergence histories averaged over 10 independent trial runs are shown in Figure 13.

6.4. DISCUSSIONS AND SUMMARIES

In cases for both practical neural network training problems studied above, it was observed that the TDE algorithm had a significantly higher convergence rate within the given limited CPU-time in comparison with the original DE algorithm. As was expected, the BP algorithm showed a rather poor performance in both these training problems.

In the case of the XOR problem, both DE algorithms completed the training task within the given maximum CPU-time, while the solution found by the BP algorithm was rather far from optimal solution (see Figure 11). As shown in Figure 11, the TDE converged to the optimal solution clearly faster than the DE, though the difference was not overwhelming.

In the Case for the aerodynamic probe calibration, because the problem was time-consuming, the training processes could not be extended long enough to allow any of the compared algorithms to converge to the global optimum. However, in this article our focus is on how the algorithms behave within the given maximum CPU-time, i.e., under tight time limitations that do not allow finding a globally

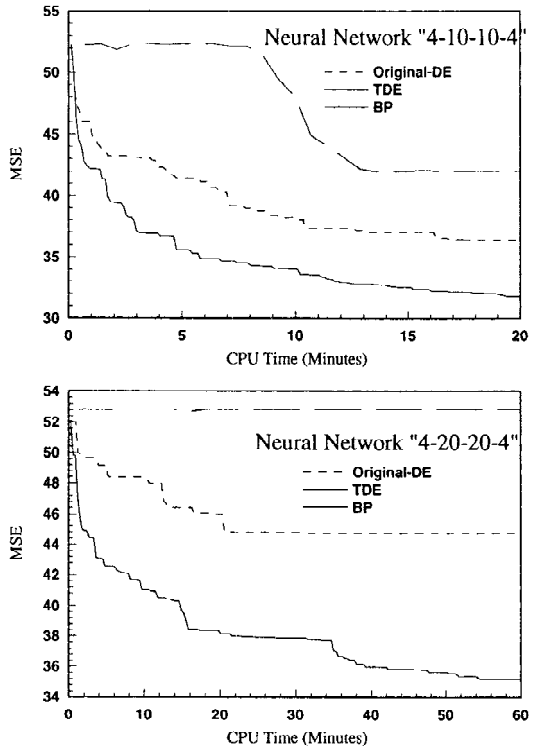

Figure 13. Convergence histories of the TDE algorithm, the original DE algorithm and the standard BP algorithm in training the neural network for aerodynamic probe calibration. Two different neural network configurations, both having two hidden layers, were studied in this case. The results are averaged over 10 independent experiments for each algorithm.

optimal solution and therefore forces us to seek the best possible sub-optimal solution that can be found within the available CPU-time. Anyway, the numerical simulation results for both studied neural network configurations, as shown in Figure 13, demonstrated a clearly higher convergence velocity with the TDE than with the original DE in the training of the neural networks to calibrate the five-hole aerodynamic probe. For both studied neural network configurations the BP algorithm, again, showed a rather poor performance by stagnating into low quality solutions at a relatively early phase in the optimization process (see Figure 13).

Concerning the trigonometric mutation probability M_t for the TDE algorithm, the setting based on the experimental results discussed in Section 5.3, $M_t = 0.05$, was found to be also suitable for both of the above-discussed neural network training problems. Note also, that all three neural network training cases discussed here, were solved by applying the same values for the TDE control parameters, F, C_r and M_t. Varying only the population size, N_p, was enough to provide all the problem specific adaptation required here.

7. Conclusions

A new mutation strategy, trigonometric mutation operation was proposed and hybridized into the DE algorithm. Since the trigonometric mutation operation is a rather greedy local search operator, this modification of the DE algorithm makes it possible to straightforwardly adjust the balance between the convergence rate and the robustness through a new introduced parameter, M_t. The greediness of the algorithm can be tuned conveniently by increasing or decreasing M_t.

The modified DE algorithm was first examined with minimization of two well-known test functions, and then further demonstrated in cases of two practical applications for training neural networks, with comparisons to the original DE and Back-Propagation algorithms. The numerical simulation results showed that when the parameter, M_t, was properly chosen, the convergence rate of the TDE algorithm could be significantly increased within a given maximum CPU-time.

It is well known that when an EA, or any stochastic search algorithm, is applied to solving a real-world problem, a trade-off has to be made between the algorithm's convergence rate and the search robustness. Often the attempts towards a higher convergence rate simultaneously increase the risk that the algorithm will prematurely converge into a local optimum. The advantage of the modified DE algorithm proposed in this paper is that it can offer a dedicated search control parameter for adjusting the balance between convergence rate and the robustness, and thereby obtain a compromise between these conflicting objectives. Consequently, the algorithm can be appropriately adjusted for a higher convergence rate without sacrificing the solution precision or search robustness too much. This kind of advantage will be especially attractive when solving real-world problems with computationally expensive objective functions, since evaluating a candidate solution for such a problem is always time-consuming. Under tight computation time limitations a robust, but slowly converging algorithm cannot provide an acceptable sub-optimal solution since the convergence rate is too low for that. In such cases a more greedy but slightly less robust algorithm appears to be the only way for obtaining better sub-optimal solutions. For these problems the modified DE algorithm appears to provide a promising alternative having a dedicated search greediness control parameter.

References

1. Bäck, T. (1996), *Evolutionary Algorithms in Theory and Practice*, Oxford University Press, Inc., Oxford, 1996.
2. De Jong, K. (1975), *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*, Ph.D. Thesis, Department of Computer and Communication Sciences, University of Michigan, Ann Arbor, MI.
3. Lampinen, J. (2000), *A bibliography of differential evolution algorithm*, Technical Report, Lappeenranta University of Technology, Department of Information Technology, Laboratory of Information Processing. Available via the Internet: http://www.lut.fi/ jlampine/debiblio.htm

4. Lampinen, J., and Zelinka, I. (1999), Mechanical engineering design optimization by differential evolution, In: Corne, D., Dorigo, M. and Glover, F. (eds), *New Ideas in Optimization*, McGraw-Hill, London (UK), pp. 127–146.
5. Lampinen, J. and Zelinka, I. (2000). On stagnation of the differential evolution algorithm, In: Ošmera, P. (ed.), *Proceedings of MENDEL 2000, 6th International Mendel Conference on Soft Computing*, Brno, Czech Republic, pp. 76–83. Available via the Internet: http://www.lut.fi/ jlampine/MEND2000.ps.
6. Hassoun, M. H. (1995), *Fundamentals of Artificial Neural Networks*, MIT Press, Cambridge, MA.
7. Muhlenbein, H., Schomisch, M. and Born, J. (1991), The parallel genetic algorithm as function optimizer, *Parallel Computing* 17, 619–632.
8. Price, K. (1996), DE: a fast and simple numerical optimizer, *1996 Biennial Conference of the North American Fuzzy Information Processing Society, NAFIPS*, Smith, M., Lee, M., Keller, J. And Yen, J. (eds.), IEEE Press, New York, pp. 524–527.
9. Price, K. (1999), An introduction to DE, In: Corne, D., Marco, D. and Glover, F. (eds.), *New Ideas in Optimization*, McGraw-Hill, London (UK), pp. 78–108.
10. Rogalsky, T., Derksen, R.W. and Kocabiyik, S. (1999), An aerodynamic design technique for optimizing fan blade spacing, *Proceedings of the 7th Annual Conference of the Computational Fluid Dynamics Society of Canada*, pp 2-29 – 2-34.
11. Rogalsky, T. and Derksen, R. W. (2000), Hybridization of differential evolution for aerodynamic design, *Proceedings of the 8th Annual Conference of the Computational Fluid Dynamics Society of Canada*, pp. 729–736.
12. Stumberger, G., Dolinar, D., Pahner, U. and Hameyer, K. (2000), Optimization of radial active magnetic bearings using the finite element technique and the differential evolution algorithm, *IEEE Transactions on Magnetics* 36(4), 1009–1013.
13. Storn, R. (1996), On the usage of differential evolution for function optimization, *1996 Biennial Conference of the North American Fuzzy Information Processing Society (NAFIPS 1996)*, Berkeley, IEEE, New York, pp. 519–523.
14. Storn, R. and Price, K. (1995), *DE-a simple and efficient adaptive scheme for global optimization over continuous space*, Technical Report TR-95-012, ICSI, March 1995. Available via the Internet: ftp.icsi.berkeley.edu/pub/techreports/ 1995/tr-95-012.ps.Z.
15. Storn, R. and Price, K. (1996), Minimizing the real function of the ICEC'96 contest by DE, *IEEE International Conference on Evolutionary Computation*, Nagoya, pp. 842–844.
16. Storn, R. and Price, K. (1997), DE-a simple evolution strategy for fast optimization, *Dr. Dobb's Journal* April 97, 18–24 and 78.
17. Storn, R. and Price, K. (1997), DE-a simple and efficient heuristic for global optimization over continuous space, *Journal of Global Optimization*, 11(4), 341–359.
18. Zaharie, D. (2002), Critical values for control parameters of differential evolution algorithms, In: Matoušek, R. and Ošmera, P. (eds.), *Proceedings of MENDEL 2002, 8th International Conference on Soft Computing*, Brno, Czech Republic, pp. 62-67. ISBN 80-214-2135-5.