A Multi-strategy Differential Evolution Algorithm for Financial Prediction with Single Multiplicative Neuron

Chukiat Worasucheep[1] and Prabhas Chongstitvatana[2]

[1] Applied Computer Science, Department of Mathematics, Faculty of Science,
King Mongkut's University of Technology Thonburi, Bangkok 10140, Thailand
chukiat.wor@kmutt.ac.th
[2] Department of Computer Engineering, Faculty of Engineering,
Chulalongkorn University, Bangkok 10330, Thailand
prabhas.c@chula.ac.th

Abstract. This paper proposes a differential evolution (DE) algorithm that combines the strengths of multiple strategies together. The selection of strategy and control parameters for each individual happens every learning period. Thus the user gains the benefits of different strategies without difficult fine tuning of control parameters. The performance of the proposed MDE algorithm is evaluated on well-known benchmark functions and is superior to some other efficient and widely used variants of DE. In addition, MDE is applied to optimize both weights and biases of a single multiplicative neuron for prediction of DJIA with 3228 samples. Experiments show its better performance than other methods in learning ability and generalization.

Keywords: Differential Evolution, Evolutionary Algorithm, Single Multiplicative Neuron, Financial Prediction.

1 Introduction

Differential Evolution (DE) algorithm is an efficient evolutionary algorithm (EA), proposed by Storn and Price, for global optimization in continuous space [1]. DE has been widely applied and shown its strengths in many application areas from scientific, engineering, to financial [2-3]. DE has three critical control parameters: crossover rate (CR), mutation factor (F), and population size (NP), that require a proper setting to ensure its performance and convergence. Different proper ranges of CR are suggested for different optimization functions. Mutation factor F usually affects the convergence speed. NP is normally allowed the user to set to handle the complexity or dimensionality of the function. In addition, DE has a number of perturbation methods or mutation strategies developed. Most of them are suitable for different characteristics of functions. While some strategies have shown good convergence for unimodal functions, others are advised for complex multimodal functions. This complex task of the strategy selection and parameter settings becomes burdens to many practitioners, and thus encourages a number of research groups to attempt to automate the chore.

This paper proposes a simple method of mixing different strategies of DE to pull out their benefits. Trigonometric mutation operator, by Fan and Lampinen [4], and

C.S. Leung, M. Lee, and J.H. Chan (Eds.): ICONIP 2009, Part II, LNCS 5864, pp. 122–130, 2009.

two other widely used strategies are first observed. The three strategies and parameter CR are self-adapted during the learning period. The proposed multi-strategy DE (or MDE) is evaluated using 5 nonlinear benchmark functions widely used in optimization literature. Then the algorithm is applied to optimize weights and biases of a single multiplicative neuron (SMN) [5] for the prediction of next-day closing index of Dow Jones Industrial Average (DJIA) during Jan 1994 to October 2006. The dataset are technical indicators computed from the five inputs: daily opening, high, low, closing indexes and the volume traded each day. The results are compared with those from back propagation and conventional DE variants.

This article is organized as follows. The next section briefly reviews basic understanding of the related topics, i.e., DE and SMN for financial prediction. Section 3 describes the proposed algorithm which is then evaluated with benchmark functions in Section 4. Section 5 describes the application in optimizing the SMN for the prediction of DJIA index. Finally Section 6 concludes this work.

2 DE and Time-Series Prediction with SMN

This section briefly reviews essential understanding of DE, its adaptation, and the financial prediction using SMN optimized with EAs.

2.1 Differential Evolution and Its Adaptation of Strategies and Parameters

Differential evolution (DE) is a population-based and directed search method [1]. Like other EAs, it starts with a randomly generated initial population of individuals, called vectors in DE domain. DE generates offspring by perturbing the solutions with a scaled difference of two randomly selected parent vectors. Then the replacement of an individual occurs only if the offspring outperforms its corresponding parent. This process will be iterated until some stopping criteria such as a maximum number of generations or number of objective function calls allowed [2]. There exist many mutation strategies in DE algorithms resulting in different variants, whose names are denoted with *DE/x/y/z* naming scheme [1-2]. The performance of the conventional DE algorithm highly depends on the chosen mutation strategy and associated parameter values (NP, CR, and F). In the past decade, DE researchers have suggested many empirical guidelines for choosing mutation strategies and tuning of the control parameter. Unfortunately various conflicting conclusions have been drawn with regard to the rules for manual configuration. Therefore researchers have developed some techniques to avoid such manual tuning. For examples, Qin et al. [6] recently proposed a self-adaptation of five conventional mutation strategies as well as parameters CR and F during the learning period. A similar work by Brest et al. [7] also encoded F and CR into each vector but randomized and adjusted them with different ranges and methods.

2.2 The Single Multiplicative Neuron for Financial Prediction

Various neural network models and training algorithms have been used for time series prediction [8-9]. Since most financial time-series are nonlinear in nature, multiplication being the most basic unit of nonlinearities has been a natural choice of models in

the artificial neuron model for financial prediction. In this work a single multiplicative neuron (SMN) [5] is employed because SMNs are easy to implement by using the standard back propagation (BP) learning algorithm and exhibit better performance than the multilayered neural networks with special structures [5,8]. The structure of a generalized SMN model with learning algorithm is briefly as follow. Let (x_1, x_2, …, x_n), (w_1, w_2, …, w_n) and (b_1, b_2, …, b_n) be the input pattern, weights and biases of the model, respectively. The output function is the *logsig* of the operator Ω which is a multiplicative operation as in Eq. (1) and u is equal to Ω.

$$y = \frac{1}{1+e^{-u}}, \text{ where } u = \Omega = \prod_{i=1}^{n}(w_i x_i + b_i). \tag{1}$$

Back propagation (BP) algorithm is generally adopted as learning algorithm to train the SMP. BP is used to minimize the error function defined in Eq. (2).

$$E = \frac{1}{2}\sum_{p=1}^{N}(y_p - d_p)^2, \tag{2}$$

where d_p represent the desired network output for the pth input pattern, and y_p is the computed output neuron. Using the steepest descent gradient approach and the chain rules for the partial derivatives, the update rules for the weights and biases of the model are given in Eqs (3) and (4), respectively.

$$w_i^{new} = w_i^{old} - \eta\frac{1}{N}\sum_{p=1}^{N}\left((y_p - d_p)y_p(1-y_p)\frac{u}{w_i x_i + b_i}x_i\right), \tag{3}$$

$$b_i^{new} = b_i^{old} - \eta\frac{1}{N}\sum_{p=1}^{N}\left((y_p - d_p)y_p(1-y_p)\frac{u}{w_i x_i + b_i}\right), \tag{4}$$

where η is the learning rate parameter that controls the convergence speed. The above process is iterative until some termination criteria are met such as the maximum generations.

Yadav et al. [5] recently applied SMN to predict a set of well-known time-series including a currency exchange rate from 2002 to 2004 (totaling of 800 samples). The result has outperformed a multilayer neuron network. Soon after that, Zhao and Yang [9] proposed a Particle Swarm Optimization as learning algorithm of SMN for three well-known time series prediction problems. Their results outperformed those from learning with BP and Genetic Algorithm.

3 The Proposed Multi-strategy Differential Evolution

This section describes the proposed multi-strategy DE (MDE) that combines the strength of various DE strategies. As discussed in Sections 1 and 2.1, different mutation strategies are suggested depending on the characteristics or complexity of the function at hands, which in most cases are usually unknown a priori. In this work, we have chosen three common strategies as follows.

1. The DE/rand/1/bin (RAND) strategy which is the most widely used strategy and regarded as the standard DE. RAND is has good exploration ability and is suitable for multimodal functions [2-3].

2. The DE/current-to-best/1/bin (CURR2BEST) strategy has demonstrated a good convergence for a wide set of functions [2, 6].
3. The trigonometric mutation operator (TMO) performs as a rather greedy local search [4]. A small value of mutation probability parameter was suggested in [4] for a general use to balance the convergence speed (of TMO) and exploration ability (of RAND) in TDE.

This set of strategies is chosen based on an idea that an efficient mix of strategies should have diverse characteristics. However, other mixes of various strategies are under investigation and are expected to provide a better performance and robustness.

The structure of DE population in MDE is as illustrated in Fig. 1 that excludes the D-dimensional position $\{x_1, x_2, \ldots, x_D\}$ and the objective function value f of each vector. For each vector i, Strategy_i indicates one of the available strategies. The current number of available strategies in this work (NS) is 3. CR_i is set for the corresponding strategy as widely suggested in DE literature i.e., 0.9 for RAND and CURR2BEST [1-3,6], and 0.95 for TMO [4]. Parameter F is related to convergence speed and most suggested to be randomized within a range of (0, 1+]. In this work, the value of F_i is randomized from a uniform distribution within [0.2, 1.1] to cover both exploration and exploitation capability.

Chance_i is a probability value determining the chance that vector i chooses one of the mutation strategies. At the beginning of evolutionary process, Chance_i is initialized equally to $1/NS$, where the current NS is 3 in this work. The roulette wheel selection method is utilized to probabilistically select one of the available strategies based on Chance_i. For the next generations, all Chance_i are recalculated every learning period λ as follows. Each vector i has an array $\text{Score}[]_i$ that continuously collects a flag indicating a success or improvement of the vector i's objective function value f_i. That is, if at generation g, vector k's f_k has improved after the mutation and crossover operation, $\text{Score}[g]_k$ is set to 1, and 0 otherwise. This array $\text{Score}[]_i$ is thus a storage for collecting or counting the number of improvement – the f of that vector i has improved – over the learning period. The size of the array $\text{Score}[]_i$ is limited to the learning period λ. Windowing scheme is used for $\text{Score}[]_i$ to let the learning mechanism able to keep only the most recent λ-generation information.

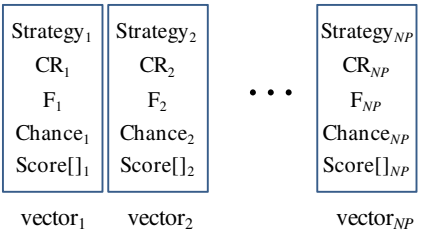

Fig. 1. The structure of the population in MDE

For any population-based search and optimization algorithms, maintaining diversity is of great importance. Some strategies to be included in MDE, such as TMO, can possess a high convergence. Therefore, the population initialization routine uniformly

randomized creates vectors of 5 times of population size (5×NP). Then one of them is randomly selected as a real population vector. Each of the remaining vectors is gradually selected into the population, only if the summation of its Euclidean distances to all vectors already in the population is of minimum. This scheme will enhance diversity at the beginning.

4 Experimentation

This section describes the experimentation to evaluate the performance of the proposed MDE algorithm by using a set of four widely-used benchmark functions for minimization. Table 1 illustrates the descriptions of those functions whose global minima are 0. The first two functions are unimodal while the remaining two functions are multimodal. The experiment will test MDE algorithm using 50 dimensions or decision variables. The maximum numbers of function calls (MAXNFC) are limited to 10000D = 500,000. The basic statistical results from executing 30 independent runs with different seed numbers for random number generator are collected. The results will be compared to those from the standard DE/rand/1/bin (CR=0.9, F=0.5) and TDE (CR=0.95, F=0.99). The associated parameter values are set as widely suggested in literature [1-3]. In all cases, the experiment investigates the results for two different values of NP, i.e. 4D and 8D. The learning period λ of MDE is set to 50 after a preliminary test of the selected functions with varying values of λ from 30 to 70 with a step of 10.

4.1 Experimental Results

Table 2 shows the means, standard deviations, the best, and the worst results from 30 runs of each algorithm for each case. Fig. 2 illustrates averages convergence graphs for each case. From the table and the figure, we can observe the following results.

1. For the simple F1 Schwefel 2.22 function, MDE clearly outperforms both RAND and TDE with both NP values tested. This demonstrates a good convergence of MDE for functions with a simple landscape.

Table 1. Search Ranges of The Minimization Problems

Function			Search Range
Schwefel 2.22	F1(x) =	$\sum_{i=1}^{D}\lvert x_i\rvert+\prod_{i=1}^{D}\lvert x_i\rvert$	[-10, 10]
Rosenbrock's	F2(x) =	$\sum_{i=1}^{D-1}\left[100(x_{i+1}-x_i^{\,2})^2+(1-x_i)^2\right]$	[-30, 30]
Rastrigin's	F3(x) =	$\sum_{i=1}^{D}\left[x_i^{\,2}-10\cos(2\pi x_i)+10\right]$	[-5.12, 5.12]
Masters (Inverted cosine wave function)	F4(x) =	$-\sum_{i=1}^{D}\left(\exp\left(\frac{-(x_i^2+x_{i+1}^2+0.5x_i\,x_{i+1})}{8}\right)\right)$ $\times\cos\left(4\sqrt{x_i^2+x_{i+1}^2+0.5x_i\,x_{i+1}}\right)+n-1$	[-5, 5]

Table 2. Statistical Results for 50 Dimensions

Function	Algorithm	NP	Mean	S.D.	Best	Worst
F1 Schwefel 2.22	RAND	200	1.87E-04	5.64E-05	9.77E-05	3.20E-04
	RAND	400	3.717	1.030	2.466	6.586
	MDE	**200**	**9.03E-19**	**3.92E-18**	**2.49E-26**	**1.80E-17**
	MDE	**400**	**3.99E-15**	**1.24E-14**	**8.88E-18**	**5.73E-14**
	TDE	200	1.71E-06	4.42E-07	1.04E-06	2.85E-06
	TDE	400	0.119	0.031	0.071	0.176
F2 Rosenbrock's	RAND	200	34.316	**0.797**	32.994	36.467
	RAND	400	47.561	**0.405**	46.860	48.346
	MDE	**200**	**1.046**	1.677	**1.67E-08**	**4.034**
	MDE	**400**	**11.913**	4.090	**3.359**	**18.948**
	TDE	200	44.537	16.791	34.678	92.721
	TDE	400	45.718	1.465	43.179	49.186
F3 Rastrigin's	RAND	200	237.031	19.692	196.060	282.849
	RAND	400	342.739	12.363	319.394	369.747
	MDE	**200**	**13.651**	**3.642**	7.960	**23.879**
	MDE	**400**	**9.711**	**3.022**	4.974	**16.914**
	TDE	200	28.560	47.051	**2.985**	141.846
	TDE	400	31.739	55.178	**2.024**	207.077
F4 Masters	RAND	200	36.821	**0.653**	35.505	37.788
	RAND	400	37.342	1.062	35.103	38.447
	MDE	**200**	**22.805**	6.817	**10.491**	**30.645**
	MDE	**400**	**30.410**	3.399	**20.515**	**32.878**
	TDE	200	35.901	1.197	31.798	37.169
	TDE	400	36.132	**0.871**	34.287	36.739

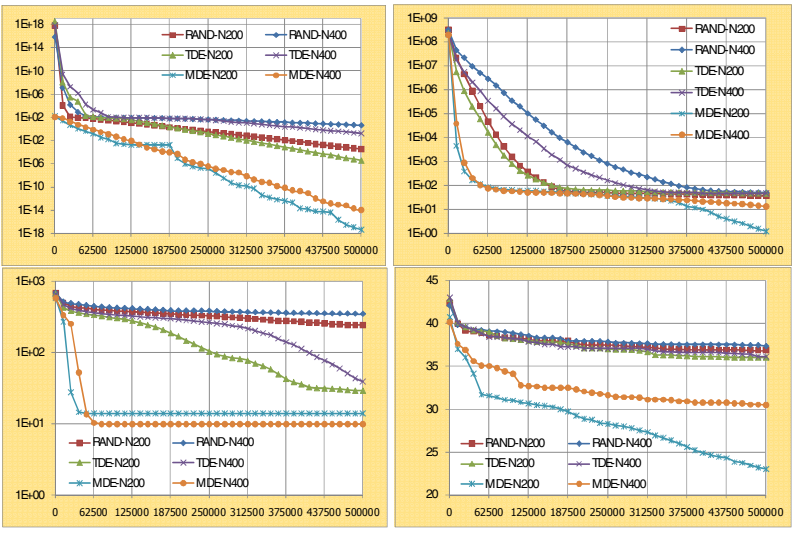

Fig. 2. Average convergence graphs: F1 Schwefel 2.22 and F2 Rosenbrock at the top; F3 Rastrigin and F4 Masters at the bottom

2. For Rosenbrock function (F2) whose global optimum is inside a long, narrow, parabolic-shaped flat valley. The convergence to the global optimum has been repeatedly used to assess the performance of optimization algorithm. MDE converges much faster than RAND and TDE in average since the beginning. In addition, MDE can successfully converge near to the optimum in some run when using NP=4D, thus resulting in the best value (at 1.67E-8) much better than all other algorithms.
3. For multimodal Rastrigin (F3) and the Masters (F4) functions whose fitness landscapes have numerous local optima due to the cosine term, MDE again outperforms other algorithms in all indicators but with only one exception. TDE can achieve the best values better than that by MDE in a few runs. However a lower SD value from MDE indicates MDE's better robustness than TDE.
4. Regarding the effects of different NP tested, in nearly all cases, all algorithms tested including MDE converge better, in average, when using NP=4D than NP=8D. The exception is only at multimodal Rastrigin case for MDE, in which a larger population tends to be more better than a smaller population. The effects of different population size and the guideline for choosing an optimal population size deserve a further investigation.

5 Financial Time-Series Prediction with SMN

This section describes an application of MDE in optimizing weights and biases of SMN during its learning, which is then used to predict the outputs during testing. The dataset is index of Dow Jones Industrial Average (DJIA) from 3 January 1994 to 23 October 2006, total samples is 3228 trading days. Raw data input are daily opening, high, low, closing, and volume traded, which are easily accessible online.

5.1 Training and Testing of the Prediction Model

The data are split into two sets – training and testing sets. The training set consists of 2510 patterns and the rest is set aside for testing [10]. In this experiment, six technical indicators reviewed of domain experts [9-10] are computed from the raw data as indicated in the Table 3 and then fed into the SMN. All the inputs are normalized to values between 0.1 and 0.9. Training of the prediction models is carried out using MDE algorithm to optimize six weights and six biases of the SMN, thus 12 decision (or objective) variables. The MDE optimization is to minimize the error, E, in eq. (2) where in this particular case, d_p and y_p are the actual and predicted indices, respectively, on day p. Due to stochastic nature of the algorithm, the optimized weight and bias values of the model are obtained through 5 independent runs with different seed number for random number generator. After each training run, the obtained set of six (weight, bias) pairs of the neuron is used for prediction of next-day closing index for the test data. The mean absolute percentage error (MAPE) in Eq (5) is used to gauge the performance of the prediction model when the testing data are used,

$$MAPE = \frac{1}{N_t} \sum_{p=1}^{N_t} \frac{\left| y_p - d_p \right|}{y_p} \times 100, \tag{5}$$

where N_t is the number of testing data. Then basic statistical values of obtained MAPEs from 5 independent runs are computed for analysis. To compare the performance of the proposed model, other two DE variants (RAND and TDE) as well as back propagation (BP) learning algorithm (in Section 2.2) are also simulated. In this experiment, the learning period λ is 50 generations. Population size (NP) is 50. Maximum objective function calls (MAXNFC) allowed to learn is 30000 for MDE, TDE, and RAND, while standard BP for SMN is allowed to learn up to the same 30000 generations. For TDE, the mutation probability (*Mp*) is set to 0.05 as suggested by the originators [4]. Learning rate η for BP is 0.7 [8-9]. Each of three DE variants is allowed to execute the same 5 independent runs and the statistical results (of MAPEs) are compared to the MAPE from the single run with SMN's BP.

Table 3. Selected technical indicators and their formula

Technical indicators used	Formula
– Exponential moving average (EMA) (3 numbers: EMA10, EMA 20, and EMA40)	(P×A)+(Prev. EMA×(1-A)); A = 2/(1+N), P is current price, A is smoothing factor, N is time period
– Accumulation/distribution oscillator (ADO)	((Close-Low) – (High-Close)) / ((High-Low) × (Period's Volume))
– Relative strength index (RSI)	RSI = 100 – 100/(1+U/D); U is total gain/n, D is total losses/n, n is number of RSI period
– Price rate of change (PROC)	$\frac{\text{(Today's close - Close X-period ago)}}{\text{Close X-period ago}} \times 100$

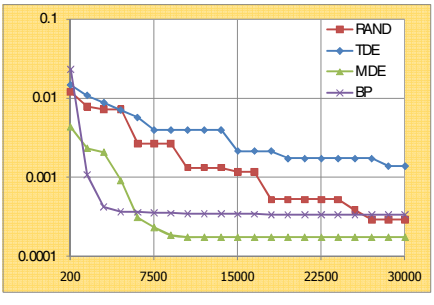

Fig. 3. Comparison of learning characteristic for DJIA from different algorithms

Table 4. Comparison of MAPE of DJIA during testing obtained from different algorithms

Algorithm	Best	Average	Worst	S.D.
RAND	1.72	22.04	74.51	30.02
TDE	19.92	31.36	34.59	6.40
MDE	1.61	1.69	1.86	0.11
BP	1.87			

5.2 Results and Discussions

The learning characteristics or convergences of four algorithms are demonstrated in Fig. 3. Table 4 compares the prediction accuracy using MAPE obtained from the algorithms. It can be observed that BP learning quickly converged and got trapped in some local optima early at MAPE=1.87. In contrary, RAND and TDE variants gradually improved their MAPEs similarly. However, RAND tends to provide a better result than TDE; this should due to the greediness of TDE that causes less exploration. A smaller value of *Mp* is expected to improve the results of TDE. Concerning the best MAPEs achieved, both RAND and MDE obtained the better values than BP.

However MDE achieved a better average MAPE than RAND. The small value of S.D. for MDE indicates its robustness in generalization for this prediction experiment.

6 Conclusion

Differential Evolution (DE) algorithm has recently gained a strong attention, and has been widely used in engineering and scientific optimization community. However its robustness and performance depend upon choosing a proper mutation strategy as well as associated control parameters. This work proposes the MDE algorithm that combines different efficient DE strategies to gain their advantages. The algorithm is evaluated using 5 well-known benchmark functions and compared to the standard DE/rand/1/bin and TDE variants. The results have demonstrated that MDE outperforms other competitive variants. Then MDE is applied to optimize both weights and biases of a single multiplicative neuron for prediction of the next-day closing index of DJIA. The results have shown that MDE can provide a better prediction in accuracy and robustness than back propagation, standard DE, and TDE. The current works are to experiment MDE with more strategies, and to evaluate with a larger set of benchmark functions and more time-series datasets.

References

1. Storn, R., Price, K.: Differential evolution—A simple and efficient heuristic for global optimization over continuous spaces. J. of Global Optimization 11, 341–359 (1997)
2. Price, K., Storn, R.M., Lampinen, J.A.: Differential Evolution: A Practical Approach to Global Optimization (Natural Computing Series). Springer, New York (2005)
3. Chakraborty, U.K. (ed.): Advances in Differential Evolution. Springer, Heidelberg (2008)
4. Fan, H.Y., Lampinen, J.: A trigonometric mutation operation to differential evolution. Journal of Global Optimization 27, 105–129 (2003)
5. Yadav, R.N., Kalra, P.K., John, J.: Time series prediction with single multiplicative neuron model. Applied Soft Computing 7, 1157–1163 (2007)
6. Qin, A.K., Huang, V.L., Suganthan, P.N.: Differential Evolution algorithm with strategy adaptation for global numerical optimization. IEEE Trans. Evol. Comp. 13, 398–417 (2009)
7. Brest, J., Greiner, S., et al.: Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems. IEEE Trans. Evol. Comp. 10, 646–657 (2006)
8. Yadav, A., Mishra, D., Yadav, R.N., Ray, S., Kalra, P.K.: Time-series prediction with single integrate-and-fire neuron. Applied Soft Computing 7, 739–745 (2007)
9. Zhao, L., Yang, Y.: PSO-based single multiplicative neuron model for time series prediction. Expert Systems with Applications 36, 2805–2812 (2009)
10. Majhi, R., Panda, G., Majhi, B., Sahoo, G.: Efficient prediction of stock market indices using adaptive bacterial foraging optimization (ABFO) and BFO based techniques. Expert Systems with Applications 36, 10097–10104 (2009)