Replicator Dynamic Inspired Differential Evolution Algorithm for Global Optimization

Shichen Liu, Yan Xiong, Qiwei Lu and Wenchao Huang
Department of Computer Science and Technology, University of Science and Technology of China, Hefei, Anhui 230027, China

Keywords: Differential Evolution, Numerical Optimization, Parameter Adaptation, Self-adaptation, Replicator Dynamic, Natural Computation.

Abstract: Differential Evolution (DE) has been shown to be a simple yet efficient evolutionary algorithm for solving optimization problems in continuous search domain. However the performance of the DE algorithm, to a great extent, depends on the selection of control parameters. In this paper, we propose a Replicator Dynamic Inspired DE algorithm (RDIDE), in which replicator dynamic, a deterministic monotone game dynamic generally used in evolutionary game theory, is introduced to the crossover operator. A new population is generated for an applicable probability distribution of the value of *Cr*, with which the parameter is evolving as the algorithm goes on and the evolution is rather succinct as well. Therefore, the end-users do not need to find a suitable parameter combination and can solve their problems more simply with our algorithm. Different from the rest of DE algorithms, by replicator dynamic, we obtain an advisable probability distribution of the parameter instead of a certain value of the parameter. Experiment based on a suite of 10 bound-constrained numerical optimization problems demonstrates that our algorithm has highly competitive performance with respect to several conventional DE and parameter adaptive DE variants. Statistics of the experiment also show that our evolution of the parameter is rational and necessary.

1 INTRODUCTION

The evolutionary algorithms are heuristic search algorithms which have been developed for over 50 years (Friedberg, 1958); (Box, 1957); (Holland, 1962); (Fogel, 1962). There are three main aspects in EAs, i.e., genetic algorithms, evolutionary programming and evolutionary strategies. They are now generally used to solve optimization problems in continuous search space. Differential Evolution (DE) algorithm, proposed by Storn and Price, is one of the state-of-the-art evolutionary algorithms (Storn and Price, 1995). DE algorithm is a simple yet efficient population-based stochastic method for global optimization problems, and it has been successfully applied to a whole host of engineering problems such as aerodynamic design (Rogalsky et al., 1999), digital filters design (Storn, 1996); (Storn, 2005), power system optimization (Lakshminarasimman and Subramanian, 2008), etc.

Generally, just as other evolutionary algorithms, there are three main operations in DE, i.e., mutation, crossover and selection. In these operations three crucial control parameters are required to be specified. They are the population size *NP*, scale factor *F* and the crossover rate *Cr*. These parameters significantly affect the optimization performance of the DE. In this regard, although the use of evolutionary algorithms to solve problems of design and optimization is varied, different end-users confront the same problem that they have to find a suitable parameter combination that matches the evolutionary algorithms before actual design or optimization can begin (Lobo and Goldberg, 2001); (Harik and Lobo, 1999). Hence, it's reasonable and necessary to turn parameters setting into a part of the algorithm itself instead of leaving it as a problem to the end-users.

In the past, on the choice of parameters of all sorts of EAs, researchers always try to find a best definite value for a parameter. To achieve this, literature either uses a trial-and-error searching process, or gets the parameter adapted or self-adapted. However, in most cases, one cannot find a best value for parameter configuration to optimize the performance, and even whether there exists such

Liu S., Xiong Y., Lu Q. and huang W..
Replicator Dynamic Inspired Differential Evolution Algorithm for Global Optimization.
DOI: 10.5220/0004053401330143
In *Proceedings of the 4th International Joint Conference on Computational Intelligence* (ECTA-2012), pages 133-143
ISBN: 978-989-8565-33-4

a best value is doubtful. More than one value of the parameter may be appropriate: a value may win in a run while a different value may perform better in another run. Furthermore, there are also many cases that some individuals of the population use a parameter while the others use a different one obtain better results than that the whole population uses a definite parameter. So, unlike the previous studies, which focus on finding a certain value for a parameter, we focus on the probability distribution of all possible or suitable values of the parameter, a definite value is just a special form of the distribution.

Based on the above observation, in this paper, we propose a Replicator Dynamic Inspired DE algorithm (RDIDE), in which crossover rate *Cr* is configured using replicator dynamic, a deterministic monotone game dynamic generally used in evolutionary game theory. Since the probability distribution of the crossover rate *Cr* is self-adapted in our algorithm, the end-users can be able to simply run the algorithm as a black-box without consideration of the parameters, which may greatly improve the working efficiency of the end-users.

To sum up, this paper makes the following contributions:

- We propose a new self-adaptive DE algorithm, with which the users can solve their problems more simply, with a higher success rate and a quicker convergence speed.
- Replicator dynamic is introduced to the parameter setting of the DE algorithm. We no more discuss about a proper parameter, but about an advisable probability distribution of the parameter.
- In the dynamic of the distribution, we design a new mechanism for believable success rate based on principle of statistics.

The remainder of this paper is organized as follows. In section 2, the conventional DE is reviewed. Section 3 describes the proposed RDIDE and the use of replicator dynamic. A suite of 10 bound-constrained numerical optimization problems is set to evaluate the performance of the algorithm in section 4. Finally, section 5 summarizes the main conclusions arising from this work.

2 DIFFERENTIAL EVOLUTION ALGORITHM

Without loss of generality, in this paper, DE is aiming to minimize an objective function. Let S be the search space of the optimization problem, $S \in R^D$. The population of DE includes *NP* individuals and each of them is a *D*-dimensional solution particle. At any certain generation *G*, the individuals are of the form $\vec{X}_i^G = (x_{i,1}^G, x_{i,2}^G, x_{i,3}^G \ldots\ldots x_{i,D}^G), \mathrm{i} = 1,2,\ldots,\mathrm{NP}$, $\vec{X}_i^G \in S$, where *i* indicates the index of the particle. The particles develop from one generation to another constraint by the search space. At each generation, every particle goes through the operations of mutation, crossover and selection, and a trial particle will be generated for each target particle. The evolution processes as follows.

2.1 Initialization

DE algorithm starts with an initial population $\{\vec{X}_1^0, \vec{X}_2^0, \vec{X}_3^0 \ldots \vec{X}_{NP}^0\}$, these particles are expected to be initialized filling the entire search space as much as possible. For this purpose, generally, the initial population is generated within the boundary constraints at random

$$X_{i,j}^0 = X_j^L + N(0,1) \times (X_j^U - X_j^L), \mathrm{j} = 1,2,\ldots,\mathrm{D} \qquad (1)$$

where X_j^L and X_j^U are the lower and upper boundary of j-th component respectively, and $N(0,1)$ denotes a uniformly distributed random value within the range [0, 1].

2.2 Mutation

A mutant vector $\vec{Y}_i^G = (y_{i,1}^G, y_{i,2}^G, y_{i,3}^G \ldots\ldots, y_{i,D}^G)$ is generated for every associated target vector $\vec{X}_i^G$ in this operation at each generation *G*. Several mutation methods could be used to generate $\vec{Y}_i^G$, and a conventional one is like this:

$$\vec{Y}_i^G = \vec{X}_{r_1^i}^G + F \times (\vec{X}_{r_2^i}^G - \vec{X}_{r_3^i}^G), \mathrm{i} = 1,2,\ldots,\mathrm{NP}, \qquad (2)$$

where index $r_1^i, r_2^i and\ r_3^i$ are random integers from the range [1, NP], mutually different, and each is different from the base index *i*. *F* is a scaling factor for differential vectors.

2.3 Crossover

Crossover operation comes after mutation. The trial vector $\vec{U}_i^G = (u_{i,1}^G, u_{i,2}^G, u_{i,3}^G, \ldots\ldots, u_{i,D}^G)$ is generated from the combination of its parent and mutant vector:

$$u_{i,j}^G = \begin{cases} \mathrm{y}_{\mathrm{i,j}}^{\mathrm{G}}, & \mathrm{if}(\mathrm{Uniform}_{\mathrm{i,j}}[0,1) < \mathrm{CR}) \mathrm{or} (\mathrm{j} = \mathrm{j}_{\mathrm{rand}}) \\ \mathrm{x}_{\mathrm{i,j}}^{\mathrm{G}}, & \mathrm{otherwise} \end{cases} \qquad (3)$$

where j_{rand} is a random index chosen from [1, D] to ensure at least one component is different from $\vec{U}_i^G$ and $\vec{X}_i^G$, and the parameter *Cr* is within the range [0, 1], indicating the crossover rate of the generation. If any component of the trial vectors is beyond the search space, they will be reinitialized randomly and uniformly within the search space.

2.4 Selection

In this phase, we determine which vector is going into the next generation and which should be deleted. The procedure is done following rule for the function minimization:

$$\vec{X}_i^{G+1} = \begin{cases} \vec{X}_i^G, & if\,(f(\vec{X}_i^G) < f(\vec{U}_i^G)) \\ \vec{U}_i^G, & otherwise \end{cases} \quad (4)$$

Every trial vector is only compared with its target vector, and the one with better fitness is kept. Hence, all the individuals of the next generation are going to get better or remain the same, thus the whole population evolves.

3 REPLICATOR DYNAMIC INSPIRED DE ALGORITHM

Being a crucial factor of the DE algorithm, control parameters selection determines the performance of the algorithm directly. Hence, a good deal of research on the parameters selection of DE has been done. Storn (1995) suggested that *F* within the range [0.5,1], *Cr* in [0.8,1] and *NP* = 5*D* or 10*D*. Gämperle et al. (2002) suggested that *NP* be between 3*D* and 8*D*, *F*= 0.6, and *Cr* between [0.3,0.9]. At the same time, several adaptive and self-adaptive mechanisms have been proposed to dynamically change the value of the parameters. Zaharie (2003) used a multipopulation method for the parameter adaptation (ADE). Omran et al. (2005) proposed a mechanism to self-adapt the scaling factor *F* (SDE). Later on, Brest et al. (2006) encoded F and *Cr* into individuals and modulate them by two parameters. In the same year, Teo (2006) proposed a DE algorithm with a dynamic population sizing strategy based on self-adaptation (DESAP). Lately, Qin et al. (2009) proposed SaDE, in which both generation strategy and the parameters are adapted.

In our paper, we focus on the adaptation of *Cr* during the evolution, as *Cr* is an especially significant parameter. The suitable choice of *Cr* can lead to good result while an improper one may result in the failure of the algorithm (Price et al., 2005).

3.1 Inspired by Replicator Dynamic

The main idea of this paper is to self-adapt the probability distribution of the crossover rate, so that the parameter could be more suitable to various kinds of problems. At the same time, different distributions of *Cr* may perform better at different generations for a certain problem, so the distribution of *Cr* is expected to be fit for every moment of the evolution as well. To achieve this, a mechanism of multiple evolutions is proposed: the first evolution refers to DE algorithm itself, and the second one means that the probability distribution of *Cr* value is evolving independently with the idea of evolutionary game theory.

We build a candidate set (CRSet), containing several possible values of *Cr*. Whenever the crossover operation is executed, each individual choose one value from the set via a particular probability distribution. The value of *Cr* is a real number within the range [0, 1], and the set is expected to cover the range uniformly. In our proposal, we let $CRSet = \{CR_1, CR_2, CR_3, CR_4, CR_5\}$, where CR_i is set to ($0.2 \times i$-0.1). For each CR_i, a P_i is assigned to indicate the probability to choose it, the distribution of P_i is $\vec{P}$. At each generation, every individual choose a *Cr* from the CRSet via the distribution of P_i, and the distribution $\vec{P}$ is evolving according to the fitness of each CR_i of the current and previous generations with replicator dynamic.

Probability distribution to choose values for *Cr* is very similar to mixed strategy equilibrium of a game theory, and a definite value of *Cr* corresponds with a pure strategy. Our attention is on the dynamically changing of the distribution, thus a method of evolutionary game theory is introduced. We assume that a new population of plentiful individuals is generated to seek a reasonable probability distribution for CR_i with the idea of evolution. Any individual in the population is called replicator, choosing a certain value in the CRSet and passing its choice to the descendants without modification. Let $n_i(t)$ be the number of individuals choosing CR_i at time point *t*, then the total population size is $N(t) = \sum_{i=1}^{5} n_i(t)$, and the proportion of individuals to choose CR_i is $pcr_i(t) = n_i(t)/N(t)$. The population state is the distribution of $pcr_i(t)$, i.e., $\vec{P}_{cr}(t) = (pcr_1(t), pcr_2(t), pcr_3(t), pcr_4(t), pcr_5(t))$, Let β and δ be the

background per capita birth and death rates in the population. Then the rate of change of the number of individuals choosing CR_i ($\dot{n}_i$) and rate of change of total population ($\dot{N}$) can be described as follows:

$$\dot{n}_i = (\beta + fitness(CR_i) - \delta) n_i \tag{5}$$

$$\begin{aligned} \dot{N} &= \sum_{i=1}^{5} \dot{n}_i = \sum_{i=1}^{5} (\beta - \delta) n_i + N \sum_{i=1}^{5} \frac{fitness(CR_i) \cdot n_i}{N} \\ &= N\left(\beta + \overline{fitness(CR)} - \delta\right) \end{aligned} \tag{6}$$

where $\overline{fitness(CR)} = \sum_{i=1}^{5} (fitness(CR_i) \cdot pcr_i)$ is the average fitness.

Since $pcr_i(t) = n_i(t) / N(t)$, we take derivative to both sides: $N \cdot \dot{pcr}_i = \dot{n}_i - \dot{N} \cdot pcr_i$. So

$$\dot{pcr}_i = \left(fitness(CR_i) - \overline{fitness(CR)}\right) \cdot pcr_i \tag{7}$$

(7) gives the replicator dynamic that will be used to adjust distribution of pcr_i, increasing rate of the proportion of the individual choosing CR_i is independent of the background per capita birth rate(β), death rate(δ) and the size($N(t)$) of the population. In another word, the evolution of $\vec{P}_{cr}$ is only dependent on the fitness of each *Cr*, which is very simple to execute. We let the possibility of one particle (an individual in DE) to choose CR_i equal the proportion of individuals (in *Cr* evolution) to choose CR_i, i.e., $P_i = pcr_i, \vec{P} = \vec{P}_{cr}$, to ensure that the proportion of individuals in DE to choose different *Cr* is approximate to $\vec{P}_{cr}$. $fitness(CR_i)$ can be indicated by the success rate of the trial vectors generated by Cr_i (CrT_i) and successfully entering the next generation $(\mathrm{CrW_i})$. So, (7) changes into the form below:

$$fitness(CR_i) = SuccRate(CR_i) \approx \frac{\mathrm{CrW_i}}{CrT_i} \tag{8}$$

$$\dot{P}_i = \left(SuccRate\,(CR_i) - \overline{SuccRate\,(CR)}\right) \cdot pcr \tag{9}$$

$$\overline{SuccRate\,(CR)} = \sum_{i=1}^{5} \left(SuccRate\,(CR_i) \cdot P_i\right), \ \mathrm{i} = 1,2,\ldots,5 \tag{10}$$

The distribution $\vec{P}$ changes by (8), (9) and (10) succinctly, thus the evolution of *Cr i*s achieved.

3.2 Design of Believable Success Rate

When we use the replicator dynamic for the evolution of *Cr*, there is still a problem in (8), that if the total quantity of individuals using CR_i is not enough, the corresponding fitness is trustless.

So we have to determine the minimum of CrT_i to ensure the trustiness of $fitness(CR_i)$ with high confidence level and narrow confidence interval. From de Moivre–Laplace theorem, we learn that if CrT_i is big, approximately

$$\frac{(CrW - CrT \cdot fitness(CR))}{\sqrt{CrT \cdot fitness(CR) \cdot (1 - fitness(CR))}} \sim \mathrm{N}(0,1)$$

where $\mathrm{N}(0,1)$ denotes the standard normal distribution and indexes are omitted. Thus we have:

$$P\left(-u_{\alpha/2} \le \frac{(CrW - CrT \cdot fitness(CR))}{\sqrt{CrT \cdot fitness(CR) \cdot (1 - fitness(CR))}} \le u_{\alpha/2}\right) \approx 1 - \alpha \tag{11}$$

Equation can be transformed into another form:

$$P(A \le fitness(CR) \le B) \approx 1 - \alpha \tag{12}$$

The solutions of the equation $\frac{(CrW - CrT \cdot fitness(CR))^2}{CrT \cdot fitness(CR) \cdot (1 - fitness(CR))} = u_{\alpha/2}^2$ are A and B:

$$A, B = \frac{CrT}{CrT + u_{\alpha/2}^2} \left[\widehat{fitness}(CR) + \frac{u_{\alpha/2}^2}{2 \cdot CrT} \pm u_{\alpha/2} \sqrt{\frac{\widehat{fitness}(CR) \cdot (1 - \widehat{fitness}(CR))}{CrT} + \frac{u_{\alpha/2}^2}{4 \cdot CrT^2}} \right] \tag{13}$$

where $\widehat{fitness}(CR) = CrW / CrT$, denotes the value of fitness that will be used in the algorithm, $u_{\alpha/2}$ is a corresponding constant to α in $\mathrm{N}(0,1)$, A takes negative sign and B takes positive sign.

$(1-\alpha)$ is the confidence level and $[A, B]$ is the confidence interval for $fitness(CR)$. Since $0 \le \widehat{fitness}(CR) \le 1$, $\widehat{fitness}(CR) \cdot (1 - \widehat{fitness}(CR)) \le 1/4$. When we assume $\widehat{fitness}(CR) \cdot (1 - \widehat{fitness}(CR)) = 1/4$, the width of $[A, B]$ equals to $u_{\alpha/2} / \sqrt{CrT + u_{\alpha/2}^2}$. Let $\alpha = 0.05$ and *the width* ≤ 0.2, we have $u_{\alpha/2} = 1.96$ and the minimum of *CrT* is 93. This is to say, *CrT* must be at least 93, $\widehat{fitness}(CR)$ is trusted.

However, *CrT* could not be big enough in one generation, so two extra methods are introduced to RDIDE. First we build memories to store the numbers of individuals choosing CR_i and those successfully entering the next generation within the last M generations. With this method, $\widehat{fitness}(CR_i) = CrW_i / CrT_i = \sum_{g=G-M+1}^{G} CrW_i^g / \sum_{g=G-M+1}^{G} CrT_i^g \ (G > M)$ where G is the current generation, and respectively, CrT_i^g and CrW_i^g denote the number of vectors

choosing CR_i in generation g and the number of those successfully entering the next generation. During the first M generations, we simply let the value of P_i be 0.2 (i = 1, 2, 3, 4, 5), and add CrT_i^g and CrW_i^g to the memories. In the following generations, P_i is dynamically changing with (9), while CrT_i^g and CrW_i^g replace CrT_i^{g-M} and CrW_i^{g-M}. Besides, we assign a constant ($P_{\min}$) to constrain the minimum of P_i, when $P_i < P_{\min}$ and P_i is going to decrease, the value of P_i remains the same. The expected value of CrT, $E(CrT) \approx NP \cdot P \cdot M$. In this paper, $E(CrT)$ is expected to be equal or greater than 100 to ensure the trustiness of $fitness(CR)$, and we achieve this by assuming a small P such as 0.1, and $M = \lceil 100/P/NP \rceil$.

3.3 The Algorithmic Description

The algorithmic description of the RDIDE is presented in Table 1.

Table 1: Algorithmic description of the RDIDE.

```
   Step 1: Initialization
     Set the generation counter G=0.
     Initialize a population of NP individuals according to (1). Evaluate the
population. Store     with best fitness as     and its fitness      .
     Initialize the distribution of   ,                , and establish two
memories,                    , (i=1 to M, j=1 to 5).
   Step 2: Evolution
     WHILE Termination Criterion is not satisfied
    Step 2.1 renovate         and
         Replace        and        by         and
         Set           ,
    Step 2.2 Mutation, Crossover and Selection
        FOR i=1 to NP
            Choose a      from CRSet due to
            FOR j=1 to D




            END FOR
            WHILE the variable is outside the search region
                 Regenerate
            END WHILE
            Evaluate the trial vector.
            IF
                                    ,
               IF


               END IF
            ELSE
            END IF
        END FOR
    Step 2.3 Dynamic Change of Distribution
        IF G>M
```

Table 2: Benchmark functions.

Test problems	$f_{\min}$	global optimum	S
$F1 = \sum_{i=1}^{D} z_i^2$	0	o	$[-100,100]^D$
$F2 = \sum_{i=1}^{D} \left(\sum_{j=1}^{i} z_j^2\right)$	0	o	$[-100,100]^D$
$F3 = \sum_{i=1}^{D-1} \left(100\left(x_i^2 - x_{i+1}\right)^2 + \left(x_i - 1\right)^2\right)$	0	$(1,1,\cdots,1)$	$[-100,100]^D$
$F4 = \left(\sum_{i=1}^{D} \left(\sum_{j=1}^{i} z_j^2\right)\right)\left(1 + 0.4\left\|N(0,1)\right\|\right)$	0	o	$[-100,100]^D$
$F5 = -20\exp\left(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D} z_i^2}\right) - \exp\left(\frac{1}{D}\sum_{i=1}^{D}\cos(2\pi z_i)\right) + 20 + e$	0	o	$[-32,32]^D$
$F6 = -20\exp\left(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D} m_i^2}\right) - \exp\left(\frac{1}{D}\sum_{i=1}^{D}\cos(2\pi m_i)\right) + 20 + e$	0	o	$[-32,32]^D$
$F7 = \sum_{i=1}^{D}\frac{z_i^2}{4000} - \prod_{i=1}^{D}\cos\left(\frac{z_i}{\sqrt{i}}\right) + 1$	0	o	$[-500,500]^D$
$F8 = \sum_{i=1}^{D}\frac{m_i^2}{4000} - \prod_{i=1}^{D}\cos\left(\frac{m_i}{\sqrt{i}}\right) + 1$	0	o	$[-500,500]^D$
$F9 = \sum_{i=1}^{D}\left(z_i^2 - 10\cos\left(2\pi z_i\right) + 10\right)$	0	o	$[-5,5]^D$
$F10 = \sum_{i=1}^{D}\left(m_i^2 - 10\cos\left(2\pi m_i\right) + 10\right)$	0	o	$[-5,5]^D$
$z = x - o, o = (o_1, o_2, \ldots, o_D), m = M(z)$;o: the shifted global optimum, M: orthogonal rotation matrix. F6:cond(M)=1; F8:cond(M)=3; F10:cond(M)=2			

4 EXPERIMENTS AND RESULTS

4.1 Test Problems and Experimental Conditions

In this paper, in order to assure a fair comparison, the experimental conditions, the parameters setting and the benchmark problems are the same to SaDE. 10 benchmark problems (F1-F10) were set to evaluate the performance of our algorithm. Six functions (F1, F2, F4, F5, F7, F9) are shifted and three (F6, F8, F10) are further rotated. Among these functions, F1-F4 are unimodal functions and F5-F10 are multimodal functions. All the functions are listed in Table 2.

In our experiment, RDIDE is compared with 5 conventional DE and 4 adaptive DE variants. In order to ensure reliability, the statistics of the experiment with these 9 DE algorithms are results found in literature (Qin et al. 2009). The conditions of the experiment are as follows:

1) Population size *NP*=50, Scaling factor *F*=0.5;
2) Dimension D=10/30 for all problems:
   FEs=100 000 with 10-*D* problems,
   FEs=300 000 with 30-*D* problems.
3) Parameters for RDIDE, $M = 20, P_{\min} = 0.1$.
4) Comparison DE, 5 conventional DE:
   DE/rand/1(*F*=0.9,*Cr*=0.1),
   DE/rand/1(*F*=0.9,*Cr*=0.9),
   DE/rand/1(*F*=0.5,*Cr*=0.3),
   DE/rand-to-best/1(*F*=0.5,*Cr*=0.3),
   DE/rand-to-best/2(*F*=0.5,*Cr*=0.3).
   4 adaptive DE variants:
   SaDE, ADE, SDE, jDE.
5) All experiments were run 50 times, independently.

4.2 Results and Analysis

1) In this section, we compare RDIDE with the 9 other DE. Two groups of comparison are conducted to show the highly competitive performance of RDIDE. In the first comparison, we concentrate on the mean and standard deviation of the functions as well as the success rates. The success rate refers to the proportion that the success runs divided by the total runs. The success of a run means that it results in a value no worse than the pre-specified optimal value, i.e., $f_{\min} + 10^{-5}$ with the number of FEs less than the pre-specified maximum number in this run. In the second comparison, we focus on the average number of function evaluations (NFE) required to find the optima, as it's a direct reflection of the convergence speed. Table 3 and Table 4 report the statistics of the first comparison, and Table 5 shows the results of the second comparison. All best results are typed in bold.

From the results of the first comparison, we can

Table 3: Results for 10-*D* problems.

Algorithm	F1			F2			F3			F4		
D=10	Mean	Std	SRate	Mean	Std	SRate	Mean	Std	SRate	Mean	Std	SRate
CDE-1	**0**	**0**	**100%**	8.89E-01	4.96E-01	0%	9.01E-01	7.94E-01	0%	2.41E+01	1.28E+01	**100%**
CDE-2	4.95E-13	5..27E-13	**100%**	1.44E-05	1.13E-05	43%	7.11E-03	2.74E-02	0%	2.42E-04	1.38E-04	**100%**
CDE-3	**0**	**0**	**100%**	9.63E-09	5.99E-09	100%	1.76E+00	1.54E+00	0%	5.42E-06	4.44E-06	83%
CDE-4	**0**	**0**	**100%**	**0**	**0**	**100%**	2.57E+00	1.86E+00	0%	**0**	**0**	**100%**
CDE-5	**0**	**0**	**100%**	9.45E-13	9.90E-13	100%	2.37E+00	2.23E+00	0%	1.04E-08	1.20E-08	**100%**
SaDE	**0**	**0**	**100%**	**0**	**0**	**100%**	**0**	**0**	**100%**	**0**	**0**	**100%**
ADE	**0**	**0**	**100%**	1.44E-04	2.48E-04	3%	1.56E+00	2.64E+00	0%	7.00E-02	5.84E-02	0%
SDE	**0**	**0**	**100%**	**0**	**0**	**100%**	2.05E+00	1.68E+00	0%	**0**	**0**	**100%**
jDE	**0**	**0**	**100%**	**0**	**0**	**100%**	1.34E-13	7.32E-13	100%	**0**	**0**	**100%**
RDIDE	**0**	**0**	**100%**	**0**	**0**	**100%**	5.78E-07	2.05E-06	100%	**0**	**0**	**100%**
Algorithm	F5			F6			F7			F8		
D=10	Mean	Std	SRate	Mean	Std	SRate	Mean	Std	SRate	Mean	Std	SRate
CDE-1	**0**	**0**	**100%**	3.81E-05	1.30E-05	90%	**0**	**0**	**100%**	1.22E-01	2.77E-02	0%
CDE-2	4.59E-07	2.41E-07	**100%**	6.86E-07	3.89E-07	100%	3.05E-01	2.02E-01	0%	2.41E-01	2.00E-01	0%
CDE-3	**0**	**0**	**100%**	3.32E-15	9.01E-16	100%	**0**	**0**	**100%**	1.60E-01	3.75E-02	0%
CDE-4	4.97E-15	1.77E-15	**100%**	4.26E-15	1.45E-15	100%	4.67E-03	8.13E-03	70%	2.91E-01	3.14E-01	0%
CDE-5	3.55E-15	1.87E-15	**100%**	3.55E-15	0	100%	**0**	**0**	**100%**	1.44E-01	3.97E-02	0%
SaDE	**0**	**0**	**100%**	**0**	**0**	**100%**	**0**	**0**	**100%**	1.37E-02	1.18E-02	20%
ADE	**0**	**0**	**100%**	**0**	**0**	**100%**	2.55E-07	1.40E-06	100%	7.93E-02	4.24E-02	0%
SDE	**0**	**0**	**100%**	**0**	**0**	**100%**	7.39E-03	7.59E-03	40%	3.81E-02	3.06E-02	0%
jDE	**0**	**0**	**100%**	**0**	**0**	**100%**	5.75E-04	2.21E-03	93%	2.26E-02	1.77E-02	7%
RDIDE	**0**	**0**	**100%**	**0**	**0**	**100%**	**0**	**0**	**100%**	**0**	**0**	**100%**
Algorithm	F9			F10			Index of test functions with 100% success rate					
D=10	Mean	Std	SRate	Mean	Std	SRate						
CDE-1	**0**	**0**	**100%**	1.33E+01	3.00E+00	0%	1, 4, 5, 7, 9					
CDE-2	8.71E+00	5.53E+00	0%	1.63E+01	2.02E-01	0%	1, 4, 5, 6					
CDE-3	**0**	**0**	**100%**	1.65E+01	2.99E+00	0%	1, 2, 5, 6, 7, 9					
CDE-4	6.63E-02	2.52E-01	93%	1.00E+01	2.32E+00	0%	1, 2, 4, 5, 6					
CDE-5	**0**	**0**	**100%**	1.63E+01	3.36E+00	0%	1, 2, 4, 5, 6, 7, 9					
SaDE	**0**	**0**	**100%**	3.80E+00	1.35E+00	0%	1, 2, 3, 4, 5, 6, 7, 9					
ADE	**0**	**0**	**100%**	9.41E+00	2.20E+00	0%	1, 5, 6, 7, 9					
SDE	6.96E-01	8.72E-01	50%	7.79E+00	3.18E+00	0%	1, 2, 4, 5, 6, 9					
jDE	**0**	**0**	**100%**	5.78E+00	3.18E+00	0%	1, 2, 3, 4, 5, 6, 9					
RDIDE	**0**	**0**	**100%**	**0**	**0**	**100%**	**1, 2, 3, 4, 5, 6, 7, 8, 9, 10**					

observe that, for 10-*D* problems, RDIDE can find the global optimal value for all test functions with 100% success rate while other DE algorithms can achieve 4-8 functions only. It outperforms most of other algorithms, and is only with a little worse mean value in F3 compared with SaDE and jDE. For 30-*D* problems, except a success rate of 90% in F3, success rates of all other functions reach 100%. At the same time, jDE succeeds in 6 functions, which is the second best of all algorithms while CDE-2, CDE-4 and ADE fail in all functions. The mean value of RDIDE in F5, F6 and F9 is a little worse than some other algorithms, yet despite all this, the corresponding mean values are 5.91E-15, 4.25E-15 and 3.30E-14 which are still very close to the optima. And in F4, F8 and F10, the results of RDIDE surpass the other algorithms completely. For both 10-*D* and 30-*D* problems, F8 and F10 are so difficult that most algorithms fail to find the global optima while RDIDE achieves with 100% success rate.

From Table 5, we can observe that the convergence speed of RDIDE is outstanding as well. For 10-D problems it holds 4 best NFE values while for 30-D problems it holds 6 best NFE values. In contrast, CDE-4 for 10-D problems and SaDE for 30-D problems, which are the second fastest from the result, holds only 2 best NFE values and 4 best NFE values respectively.

2) The main idea of the proposed algorithm is to self-adapt the crossover rate, which is reflected by

Table 4: Results for 30-*D* problems.

Algorithm	F1			F2			F3			F4		
D=30	Mean	Std	SRate	Mean	Std	SRate	Mean	Std	SRate	Mean	Std	SRate
CDE-1	**0**	**0**	**100%**	3.62E+03	8.22E+02	0%	3.39E+01	1.51E+01	0%	1.24E+04	2.15E+03	0%
CDE-2	4.50E-02	6.15E-02	0%	1.73E+03	1.40E+03	0%	1.04E+02	6.25E+01	0%	8.98E+03	5.74E+03	0%
CDE-3	**0**	**0**	**100%**	1.38E+03	2.53E+02	0%	2.14E+01	1.98E+00	0%	5.19E+03	1.24E+03	0%
CDE-4	1.90E-07	1.04E-06	97%	1.92E+01	2.27E+01	0%	6.37E+01	4.01E+01	0%	2.36E+00	5.47E+00	0%
CDE-5	**0**	**0**	**100%**	1.04E+02	8.25E+01	0%	2.08E+01	1.19E+01	0%	1.51E+03	2.07E+02	0%
SaDE	**0**	**0**	**100%**	**0**	**0**	**100%**	3.99E-01	1.22E+00	90%	3.37E+00	1.37E+01	0%
ADE	**0**	**0**	**100%**	3.04E+02	6.86E+01	0%	4.69E+01	2.64E+01	0%	6.75E+04	1.02E+04	0%
SDE	4.56E-01	2.08E+00	50%	1.58E+00	4.48E+00	0%	7.73E+03	3.27E+04	0%	4.67E+02	5.09E+02	0%
jDE	**0**	**0**	**100%**	8.91E-11	1.27E-10	100%	5.57E-01	1.38E-00	40%	2.15E-01	4.91E-01	0%
RDIDE	**0**	**0**	**100%**	**0**	**0**	**100%**	**2.90E-01**	**2.04E+00**	**90%**	**3.85E-09**	**1.50E-08**	**100%**
Algorithm	F5			F6			F7			F8		
D=30	Mean	Std	SRate	Mean	Std	SRate	Mean	Std	SRate	Mean	Std	SRate
CDE-1	**0**	**0**	**100%**	3.81E-05	1.30E-05	90%	**0**	**0**	**100%**	9.12E-02	3.08E-02	0%
CDE-2	3.86E-02	2.18E-02	0%	7.64E-02	5.11E-02	0%	1.52E-01	1.15E-01	0%	9.01E-01	1.40E-01	0%
CDE-3	4.03E-15	1.23E-15	100%	3.67E-15	6.49E-16	100%	**0**	**0**	**100%**	2.24E-05	1.19E-04	93%
CDE-4	3.10E-02	1.76E-01	93%	4.82E-03	2.64E-02	97%	1.08E+01	1.00E+01	0%	1.82E+02	5.47E+01	0%
CDE-5	7.58E-15	1.80E-15	100%	7.34E-15	1.30E-15	100%	**0**	**0**	**100%**	3.97E-03	1.85E-02	37%
SaDE	**0**	**0**	**100%**	**0**	**0**	**100%**	2.38E-03	5.03E-03	80%	8.54E-03	0.09E-03	40%
ADE	**0**	**0**	**100%**	**0**	**0**	**100%**	**0**	**0**	**100%**	2.93E-03	5.65E-03	10%
SDE	2.19E-01	3.87E-01	40%	1.01E-01	3.04E-01	63%	1.59E+00	2.23E+00	13%	1.39E+00	4.24E+00	13%
jDE	**0**	**0**	**100%**	**0**	**0**	**100%**	**0**	**0**	**100%**	5.17E-03	6.64E-03	57%
RDIDE	5.91E-15	1.79E-15	100%	4.25E-15	8.52E-16	100%	**0**	**0**	**100%**	**0**	**0**	**100%**
Algorithm	F9			F10			Index of test functions with 100% success rate					
D=30	Mean	Std	SRate	Mean	Std	SRate						
CDE-1	**0**	**0**	**100%**	1.68E+02	1.43E+01	0%	1, 5, 7, 9					
CDE-2	8.54E+01	3.30E+01	0%	2.45E+02	2.20E+01	0%	None					
CDE-3	3.10E+01	3.24E+00	100%	1.87E+02	1.09E+01	0%	1, 5, 6, 7, 9					
CDE-4	9.58E+00	3.88E+00	93%	1.44E+02	2.09E+01	0%	None					
CDE-5	4.03E+01	3.73E+00	100%	1.88E+02	7.15E+00	0%	1, 5, 6, 7, 9					
SaDE	**0**	**0**	**100%**	1.67E+01	5.26E+00	0%	1, 2, 5, 6, 9					
ADE	2.32E-01	5.01E-01	100%	1.21E+02	1.28E+01	0%	None					
SDE	1.09E+01	4.23E+00	50%	3.63E+01	6.78E+00	0%	1, 5, 6, 7, 9					
jDE	**0**	**0**	**100%**	3.65E+01	8.29E+00	0%	1, 2, 5, 6, 7, 9					
RDIDE	3.30E-14	2.83E-14	100%	**3.07E-14**	**2.86E-14**	**100%**	**1, 2, 4, 5, 6, 7, 8, 9, 10**					

the dynamically change of the distribution of the probabilities to choose different *Cr* value. So we discuss about the property of RDIDE via the changes of the distributions in this section. Figure 1 illustrates changes of $\bar{P}$ in RDIDE for all functions with both D=10 and D=30. In the Figure, x-axis represents different values of *Cr*, y-axis represents generations of the algorithm and z-axis represents the probabilities to choose different values of *Cr*.

From the figure, it can be observed that the distribution is evolving as the DE algorithm goes on. Different values of *Cr* are suitable for different problems, and generally, a proper choice of *Cr* value is 0.1 and 0.9. Besides, even for a certain problem, the proper probability distribution of *Cr* value may change with the process of the algorithm. And we discover that this kind of change is regular, as for each problem, experiment was run 50 times independently, and the corresponding changes of the distribution are exceedingly similar. In F1, F7 and F8, *Cr* should be constant 0.1, and in F2, *Cr* should be constant 0.9. However in all other test problems, the distribution should be changing as the algorithm goes on, e. g., in F3 with D=30, the value of *Cr* should be 0.1 with high probability at the beginning of evolution, then it should change to 0.9 and be back to 0.1 finally; in F5 with both D=10 and D=30, the change of the distribution is complex at the beginning, each value of *Cr* dominates for a short time and 0.9 turns into the best choice finally; case

Table 5: Comparison of NFE.

10D	CDE-1 NFE SRate	CDE-2 NFE SRate	CDE-3 NFE SRate	CDE-4 NFE SRate	CDE-5 NFE SRate	SaDE NFE SRate	RDIDE NFE SRate
F1	16770 100%	53298 100%	10291 100%	**6318 100%**	10058 100%	8357 100%	10370 100%
F2	-- 0%	-- 43%	72436 100%	23383 100%	53658 100%	14867 100%	**14847 100%**
F3	-- 0%	-- 0%	-- 0%	-- 0%	-- 0%	**42446 100%**	64525 100%
F4	-- 0%	-- 0%	-- 83%	30925 0%	71278 100%	15754 100%	**9433 100%**
F5	25335 100%	82919 100%	15157 100%	**9436 100%**	15045 100%	12123 100%	12729 100%
F6	-- 90%	85272 100%	16682 100%	**9923 100%**	16980 100%	12244 100%	15794 100%
F7	41247 100%	-- 0%	**29961 100%**	-- 70%	59205 100%	35393 100%	54942 100%
F8	-- 0%	-- 0%	-- 0%	-- 0%	-- 0%	-- 0%	**54561 100%**
F9	**19200 100%**	-- 0%	23155 100%	-- 93%	30621 100%	23799 100%	26007 100%
F10	-- 0%	-- 0%	-- 0%	-- 0%	-- 0%	-- 0%	**24288 100%**
30D							
F1	66339 100%	-- 0%	34687 100%	-- 97%	31470 100%	**20184 100%**	32346 100%
F2	-- 0%	-- 0%	-- 0%	-- 0%	-- 0%	118743 100%	**117799 100%**
F3	-- 0%	-- 0%	-- 0%	-- 0%	-- 0%	-- 90%	**-- 90%**
F4	-- 0%	-- 0%	-- 0%	-- 0%	-- 0%	-- 0%	**191469 100%**
F5	92941 100%	-- 0%	49822 100%	-- 93%	45948 100%	**26953 100%**	28594 100%
F6	-- 0%	-- 0%	55108 100%	-- 97%	49961 100%	**33014 100%**	46740 100%
F7	80741 100%	-- 0%	39436 100%	-- 0%	41314 100%	-- 80%	**39056 100%**
F8	-- 0%	-- 0%	-- 0%	-- 0%	-- 0%	-- 0%	**45708 100%**
F9	90391 100%	-- 0%	-- 0%	-- 0%	-- 0%	**58732 100%**	147483 100%
F10	-- 0%	-- 0%	-- 0%	-- 0%	-- 0%	-- 0%	**134735 100%**

in F6 is similar to F5, yet 0.1 takes a more dominating place initially. From later period in F9 and F10 with D=10, we see that probability of 0.1 and probability of 0.9 are equal, neither of the value can surpass the other one.

So we conclude that an appropriate probability distribution of the value of *Cr* is not only related to the problem and the algorithm, but also the stage of the evolution as well. Thus assuming a constant value of *Cr* in conventional DE is not befitting, and so does using a trial-and-error process to find the parameter combination. Based on the analysis above, RDIDE, which uses the probability distribution instead of a definite value while the distribution is self-adapted, is more rational for global optimization.

5 CONCLUSIONS

In this paper, to make DE algorithm more practical to various kinds of optimization, we proposed a RDIDE algorithm, in which replicator dynamic is introduced to the crossover operator. With this method, the end-users can simply run the algorithm without considering the setting of the parameters. The algorithm involves multiple evolutions: the first evolution refers to DE algorithm, and the second one means that the parameter *Cr* is evolving independently with replicator dynamic. A new population is assumed to find an advisable probability distribution of *Cr,* and an extra technique is designed for a believable success rate. The final process according to the evolution is rather succinct.

We then compare RDIDE with 9 other DE algorithms over a suite of 10 bound-constrained numerical optimization problems and RDIDE produced highly competitive results in both success rate and the convergence speed. Furthermore, the statistics of the experiment show that a good choice of *Cr* not only rests with different problems but also with different stages of the detailed evolution process. Finally we conclude that RDIDE is a more effective and simple DE algorithm to obtain the global optima with a higher success rate and a quicker convergence speed.

ACKNOWLEDGEMENTS

This work is supported by National Natural Science Foundation of China under grant No.61170233, No.60970128, post-doctoral foundation No.2011M501397 and youth foundation of USTC. We thank four anonymous referees for their precious comments to improve this paper.

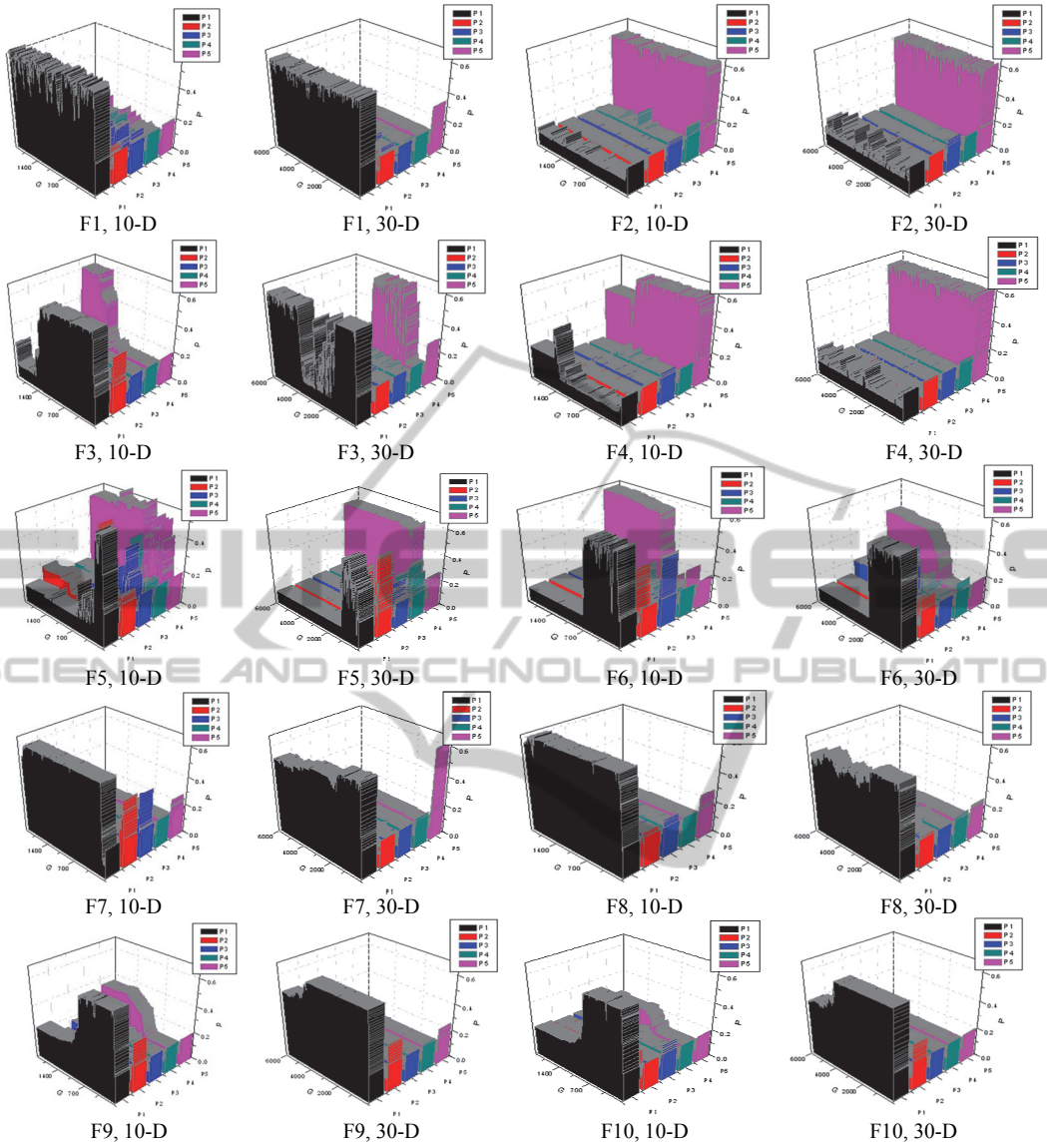

Figure 1: Dynamic change of distributions of $\vec{P}$.

REFERENCES

Friedberg, R. M., (1958). A learning machine: Part I. IBM *Journal of Research and Development*, 2, 2–13.

Box, G. E. P., (1957). Evolutionary operation: A method for increasing industrial productivity. *Applied Statistics*, 6, 81–101.

Holland, J. H., (1962). Outline for a logical theory of adaptive systems. *Journal of the Association for Computing Machinery*, 3, 297–314.

Fogel, L. J., (1962). Autonomous automata. *Industrial Research*, 4, 14–19.

Storn, R. and Price, K., (1995). Differential evolution: a simple and efficient adaptive scheme for global optimization over continuous spaces. *Technical Report TR-95-012, International Computer Science Institute*, Berkeley.

Storn, R., (1996). Differential evolution design of an IIR-filter. In *Proceedings of IEEE International Conference on Evolutionary Computation* (pp. 268–273).

Storn, R., (2005). Designing nonstandard filters with differential evolution. *IEEE Signal Processing Magazine*, 22, 103–106.

Lakshminarasimman, L. and Subramanian, S., (2008). Applications of differential evolution in power system optimization. *Studies in Computational Intelligence*, 143, 257-273.

Lobo, F. G. and Goldberg, D. E., (2001). The parameter-less genetic algorithm in practice. *Technical Report 2001022, University of Illinois at Urbana-Champaign*, Urbana, IL.

Harik, G. R. and Lobo, F. G. (1999). A parameter-less genetic algorithm. In Banzhaf et al. (Eds.), *Proceedings of the 1999 genetic and evolutionary computation conference* (vol. 1, pp. 258–265.), Morgan Kaufmann: Orlando.

Gämperle, R., Müller, S. D. and Koumoutsakos, P., (2002). A parameter study for differential evolution. In Grmela, and Mastorakis (Eds.), Advances in Intelligent Systems, Fuzzy Systems, Evolutionary Computation (pp. 293–298*). WSEAS Press: Interlaken*.

Omran, M. G. H., Salman, A. and Engelbrecht, A. P., (2005). Self-adaptive differential evolution. In *Lecture Notes in Artificial Intelligence* (pp. 192–199), Springer-Verlag: Berlin.

Brest, J., Greiner, S., Boskovic, B., Mernik, M. and Zumer, V., (2006). Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems. *IEEE Transactions on Evolutionary Computation*, 10, 646–657.

Teo, J., (2006). Exploring dynamic self-adaptive populations in differential evolution. *Soft Computing-A Fusion of Foundations, Methodologies and Applications*, 10, 637–686.

Qin, A. K., Huang, V. L, and Suganthan, P. N., (2009). Differential Evolution Algorithm With Strategy Adaptation for Global Numerical Optimization. *IEEE Transactions on Evolutionary Computation*, 13, 398 – 417.

Price, K., Storn, R., and Lampinen, J., (2005). *Differential Evolution—A Practical Approach to Global Optimization.* Springer-Verlag: New York.

Rogalsky, T., Derksen, R. W., and Kocabiyik, S., (1999). Differential evolutionin aerodynamic optimization. In *Proceedings of the 46th Annual Conference of the Canadian Aeronautics and Space Institute* (pp. 29–36), Montreal.

Zaharie, D., (2003). Control of population diversity and adaptation in differential evolution algorithms. In Matousek and Osmera (Eds.), *Proceedings of 9th International Conference on Soft Computing* (pp. 41–46), Brno.