Differential Evolution vs. the Functions of the 2nd ICEO

Kenneth V. Price
836 Owl Circle
Vacaville, Ca. 95687
e-mail: kprice@solano.community.net

Abstract - **Differential Evolution (DE) is a simple evolutionary algorithm for numerical optimization whose most novel feature is that it mutates vectors by adding weighted, random vector differentials to them. A new version of the DE algorithm is described and the results of its attempts to optimize the 7 real-valued functions of the 2nd ICEO are tabulated. DE succeeded in finding each function's global minimum, although the number of evaluations needed in one instance was unacceptably high. Despite this lone difficulty, DE's speed of execution across the remaining test bed, in addition to its simplicity, robustness and ease of use, suggest that it is a valuable tool for continuous numerical optimization.**

I. Introduction

One of the reasons for establishing the 1st International Competition on Evolutionary Optimization [1] was to provide an unbiased comparison of competing algorithms. In addition, it was hoped that new and improved algorithms would emerge from the competition. In line with these aspirations, Differential Evolution (DE) [2] has evolved in an effort to improve upon its third place finish in last year's 1st ICEO. A refinement of the procedure by which trial vectors are generated has resulted in an algorithm that not only preserves the simplicity of the original DE approach, but also yields better performance across the 1st and 2nd ICEO test beds than does last year's DE contender.

After examining the motivations for the design decisions upon which DE is based, a detailed description of the new algorithm is provided, including a short pseudo-code listing in C. Tables summarizing DE's performance on the 2nd ICEO test bed are then presented, followed by a brief discussion of the results.

II. History and Motivation

Differential Evolution grew out the author's attempts to use Genetic Annealing [3] to solve the Tchebychev Polynomial fitting problem that had been posed to him by Dr. R. Storn. Briefly, Genetic Annealing is a genetic algorithm that uses a thermodynamic annealing criterion to determine whether or not newly generated configurations should be accepted. Annealing is implemented *via* acceptance thresholds whose values are driven by a simple adaptive mechanism. Despite the success of Genetic Annealing at a variety of combinatorial tasks, the problem of finding the Tchebychev polynomials proved exceedingly difficult. Only after many hours of tuning the algorithm's control variables was a solution possible. Even once an effective set of control constants was in place, the function's minimum still required hours of computing time to resolve.

Careful consideration of the unique difficulties posed by continuous optimization, coupled with a desire to make mutation an adaptive procedure, led to replacing Genetic Annealing's traditional bit-inversion mutation scheme with a method that perturbs real-valued vectors with population-derived noise. Once the new mutation scheme was in place, the Tchebychev problem could be solved in seconds. While many aspects of DE's crossover and selection procedures still resemble their original Genetic Annealing counterparts, DE has never benefited from the use of an annealing criterion. Consequently, current versions of DE do not provide an annealing option, although the Genetic Annealing algorithm offers a straightforward implementation should it ever be required.

This year's modifications to DE reflect the inability of last year's DE algorithm to competitively solve two functions in the 1st ICEO test bed. This improvement comes without significantly degrading DE's performance on the remaining test bed functions.

III. Differential Evolution

A. Overview

The Differential Evolution algorithm is controlled by just three variables: the population size, N, the mutation scaling factor, F, and the crossover constant, CR. The population consists of N, real-valued, D-dimensional vectors whose initial parameter values are randomly chosen from within bounds set by the user. Unless otherwise specified, a uniform random distribution is used to make all decisions referred to herein as "random". Once every generation, each vector in the population becomes a *target* vector. Each target vector is combined with a *donor* vector and a *random vector differential* in order to produce a *trial* vector. If the cost of the trial vector is less than or equal to the cost of the target, the trial vector replaces the target vector in the next generation. We now examine each element of the DE algorithm in greater detail.

B. Initialization

In order to execute DE in parallel, two NxD arrays must be declared. The primary array, X1[N][D], holds the current vector population while the secondary array, X2[N][D], is used as a workspace for constructing trial vectors. Primary array vectors are initialized by assigning each parameter in every vector a randomly chosen value from within a user-defined range. Initial parameter ranges should be large enough to contain the suspected minimum, although DE does very well at locating minima that lie outside the initial bounding region. Once each vector has been initialized, its cost is evaluated and stored for future reference.

C. Mutation

Mutation generally refers an operation that adds a zero-mean random variable to one or more vector parameters. Unlike Evolutionary Strategies [4], DE does not use a predefined (e.g., Gaussian, Cauchy, fuzzy) probability density function to generate perturbing fluctuations. Instead, DE relies upon the population itself to supply increments of the appropriate magnitude and orientation. To generate a mutation increment, DE randomly selects two population vectors, forms their difference, then scales the result by F. Equation 1 shows how two randomly selected primary array vectors, $\underline{X1}b$ and $\underline{X1}c$, are used to mutate the vector, $\underline{X1}a$, into $\underline{X1}a^*$.

$$\underline{X1}a^* = \underline{X1}a + F * (\underline{X1}b - \underline{X1}c) \qquad (1)$$

Experiments with previous versions of DE suggested that F lies in the range: (0, 2). In the context of the new version of DE, however, it appears that F belongs to the interval: (-1,1). In the event that mutation causes a parameter to exceed a bound that the user has deemed to be constrained, then either the boundary point itself, or the point midway between the exceeded limit and the parameter's prior value ought be used instead.

Using randomly sampled vector differentials for the purpose of mutation has several advantages. First of all, the mean of the distribution of such differentials is always zero, since $\underline{X1}b$-$\underline{X1}c$ and $\underline{X1}c$-$\underline{X1}b$ occur with equal frequency in a random sample. Consequently, the population will not drift as a result of a sampling bias. Another advantage is that the standard deviation in each dimension of the differential's distribution adapts to the changing size and shape of the population in solution space. This self-referential property appears to be especially valuable for problems, like the Tchebychev polynomial, whose parameters exhibit vastly different ranges and sensitivities. Furthermore, this same scheme works well as a local optimizer because the differentials generated by a converging population eventually become infinitesimal. Consequently, no resolution limits need to be set. In successful optimizations, differentials shrink until the precision of the user's floating-point format is exhausted. Finally, DE does not require the user to choose a separate mutation probability. As the next section illustrates, the crossover constant, CR, effectively determines when a parameter should be mutated.

D. Crossover

One of the differences between last year's and this year's versions of DE is the way in which donor vectors are chosen. In the original version of DE, the donor is simply a randomly chosen population vector. This choice produces a very robust search, but execution is slower than it needs to be because no preference is given to low-cost donors. At the other extreme, employing the best-so-far vector as a donor [5] often speeds execution, but premature convergence can be a problem. The present incarnation of DE takes the middle path by accepting as a donor any randomly selected vector whose cost is less than or equal to that of its target. This scheme preferentially chooses low-cost vectors as donors, without draining the population of its diversity. Although a more efficient selection method for donors no doubt exists, DE chooses potential donors from the primary array at random until one meeting the lower-cost criterion is found. In the event that the target is also the lowest-cost member of the population, the target becomes its own donor.

Once a donor , $\underline{X1}d$, has been found whose cost is less than or equal to that of its target, $\underline{X1}i$, the two vectors are combined along with a scaled differential, $F*(\underline{X1}b-\underline{X1}c)$, according to the following prescription to produce a mutant vector, $\underline{X1}m$:

$$\underline{X1}m=(F + .5)*\underline{X1}d + (F - .5)*\underline{X1}i + F*(\underline{X1}b - \underline{X1}c) \qquad (2a)$$

or, alternatively:

$$\underline{X1}m= \underline{X1}d + \underline{X1}i)/2 + F*(\underline{X1}d - \underline{X1}i + \underline{X1}b - \underline{X1}c) \qquad (2b)$$

The addition of the target vector to this formula constitutes the other modification to last year's version of DE. Observe that when F = .5, the target vector, $\underline{X1}i$, drops out and equation 2 reduces to the original DE formulation:

$$\underline{X1}m = \underline{X1}d + F^*(\underline{X1}b - \underline{X1}c), \qquad F = .5 \qquad (3a)$$

On the other hand, when F = -.5, the donor vector, $\underline{X1}d$, drops out, leaving the mutation increment to be added to the target directly:

$$\underline{X1}m = \underline{X1}i + F^*(\underline{X1}b - \underline{X1}c), \qquad F = -.5 \qquad (3b)$$

Once the donor, target, and differential vectors have been combined according to equation 2, parameters from the resulting vector, $\underline{X1}m$, are crossed with those of the target vector, $\underline{X1}i$, to create a trial vector in the secondary array. Which parent vector contributes which parameter is determined by a series of D-1 binomial experiments. Only D-1 decisions are needed to construct a trial vector with D parameters, because the trial vector always takes one parameter from $\underline{X1}m$ to ensure that no effort is wasted during selection comparing the target to itself.

The D-1 binomial experiments that decide the composition of the trial vector are mediated by the crossover constant, CR, where $CR \in [0,1]$. The source of each trial vector parameter is determined by comparing CR to a uniformly distributed random number from the interval: [0,1). If the random number is greater than CR, the trial vector parameter comes from the target; otherwise, the parameter is taken from $\underline{X1}m$. In the special case where CR=1, the trial vector becomes an exact replica of $\underline{X1}m$. By contrast, the trial vector becomes a copy of the target vector when CR=0, except for the one parameter that $\underline{X1}m$ is compelled to contribute. Once the composition of the trial vector has been determined, its cost is evaluated and the decision of whether or not to accept it can be made.

E. Selection

In DE, the process of determining which trial vectors ought to survive into the next generation is very straightforward. At the end of each generation, any trial vector whose cost is less than or equal to that of its parent target vector replaces that parent in the primary array.

IV. Pseudo-code

Below is a C-style routine implementing the new DE algorithm.

Initialization:
```
for ( i=0; i=N; i++ )                        /* population loop */
{                                     /* hi, lo = parameter limits */
  for (j=0; j<D; j++)                            /* vector loop */
  {
    x1[i][j]=lo+rnd1( )*(hi-lo);         /* 0 <= rnd1( ) <1 */
  }
  cost[i]=evaluate(x1[i][0], D);         /* cost[i] = f(X1i) */
}
```

Main Loop:
```
for (g=1; g<=max_gens; g++)              /* generation loop */
{                                          /* build N trial vectors */
  for (i=0; i<N; i++)                  /* population loop, N>3 */
  {                /* pick donor and differential vectors */
    do d=rnd1( )*N; while (cost[d]<cost[i]);
    do b=rnd1( )*N; while (b==i || b==d);
    do c=rnd1( )*N; while (c==i || c==d || c==b);
    jr=rnd1( )*D;        /* pick a parameter at random */
    for (j=0; j<D; j++)     /* build a trial vector in X2i */
    {                       /* perform D-1 binomial trials */
      if (rnd1()<=CR || j==jr)
      {          /* j-th trial vector parameter is either: */
        diff=F*(x1[d][j]-x1[i][j]+x1[b][j]-x1[c][j]);
        x2[i][j]=(x1[i][j]+x1[d][j])/2+diff;
      }                /* or else j-th trial parameter is: */
      else x2[i][j]=x1[i][j];
    }
  }
  for (i=0; i<N; i++)                    /* select trial vectors */
  {                        /* compute cost of trial vector */
    score=evaluate(x2[i][0],D);
    if (score<=cost[i])                   /* compare to target */
    {                             /* lower-cost trial vectors... */
      cost[i]=score;               /* replace their targets */
      for (j=0; j<D; j++) x1[i][j]=x2[i][j]);
    }
  }                                      /* do next generation */
}                                               /* end of program */
```

V. Test Bed Results

This year's 2nd ICEO test bed consists of 7, real-valued functions, one of which is subject to a pair of constraints. Each function occurs twice in the test bed so that it can be evaluated at two different dimensions. In response to the lessons learned from last year's competition, new criteria for measuring algorithmic performance have been introduced. To gauge an algorithm's raw speed as well as its effectiveness as a local optimizer, each contestant has been asked to report the average best value reached by their algorithm at each of 5 checkpoints. The initial checkpoint for each instance of every function is given, and each subsequent checkpoint is twice as large as the previous one. Averages are computed for twenty independent trials and a trial is considered to be over when the final checkpoint is reached. In addition to reporting the average best values at each checkpoint, the very best value found (BVAT) during any of the twenty trials must also be provided. For some functions, a value-to-reach (VTR) is

Table 1 "N/A" = not applicable. "-" = did not converge within allowed limits, hence ENES is undefined.

Function	D	CP1	VTR	Mean Best Values sampled at CP1*M evaluations					BVAT	ENES
				M=1	M=2	M=4	M=8	M=16		
Generalized Rosenbrock	5	1000	1.0e-6	7.151e+0	8.771e-1	7.406e-6	VTR	VTR	0.00	4561
	10	5000	1.0e-6	3.054e+1	4.844e+0	5.895e-2	VTR	VTR	0.00	27169
Odd Square	5	1000	-.999	-3.859e-1	-6.661e-1	-8.405e-1	-8.743e-1	-9.235e-1	-1.149e+0	36395
	10	5000	-.999	-4.145e-1	-7.349e-1	-8.431e-1	-8.585e-1	-8.700e-1	-8.732e-1	-
Modified Langerman	5	650	N/A	-3.440e-1	-5.252e-1	-6.894e-1	-9.359e-1	-9.625e-1	-9.650e-1	N/A
	10	3750	N/A	-3.483e-1	-1.762e-1	-4.597e-1	-6.871e-1	-8.343e-1	-9.650e-1	N/A
Shekel's Foxholes	5	4000	N/A	-2.72e+0	-4.21e+0	-7.56e+0	-9.65e+0	-1.02e+1	-1.040e+1	N/A
	10	16000	N/A	-9.67e-1	-2.67e+0	-7.50e+0	-1.01e+1	-1.02e+1	-1.028e+1	N/A
Epistatic Michalewicz	5	312	-4.687	-2.80e+0	-3.35e+0	-3.95e+0	-4.56e+0	-4.67e+0	-4.826e+0	5339
	10	1250	-9.66	-5.77e+0	-6.79e+0	-7.81e+0	-8.90e+0	-9.44e+0	-9.660e+0	77365
Tchebychev Polynomial	9	1500	1.0e-6	5.88e+4	4.12e+3	4.64e+1	2.23e-2	VTR	0.00	17325
	17	10000	1.0e-6	4.91e+6	3.54e+4	1.18E+1	3.85e-5	VTR	0.00	78937
The Bump	10	1000	N/A	1.01e-3	1.48e-5	1.57e-9	2.59e-17	1.32e-29	4.37e-29	N/A
	20	7500	N/A	3.82e-5	3.99e-9	2.68e-15	1.25e-24	5.77e-29	5.35e-29	N/A

defined such that any vector whose cost is less than or equal to the VTR is presumed to lie within a basin of attraction that contains a global minimum. For the purpose of computing the average best values indexes, results less than the VTR are to be treated as equal to it. Consequently, no best value index can exceed the VTR if one is provided, although the BVAT often does. The total number of function evaluations over twenty trials, divided by the number of trials that successfully attain the VTR is called the ENES. As the expected number of function evaluations needed to reach the VTR, the ENES measures both the speed and consistency with which the VTR is reached. Table 1 summarizes DE's performance as measured by these indexes.

This year's competition introduces a new index, designed by the author, aimed at quantifying the effort needed to "tune" an algorithm's control parameters. Based on the concept of entropy, the "crafting effort" is a rough measure the volume of the space one must expect to search in order to find a set of control parameters that can quickly minimize an arbitrary function. More specifically, the crafting effort is the entropy of the control parameter settings required by an algorithm to solve the 14 test bed functions. Instead of using the population size, N, as a variable n the entropy computation, the ratio N/D was used instead. This choice reflects confidence in the rule that in general, one ought to use an initial population of 10

Table 2 Crafting effort

Function	Dim	DE Settings		
		N/D	F	CR
Generalized Rosenbrock	5	6.0	.6	1.0
	10	6.0	.6	1.0
Odd Square	5	10.0	.5	.8
	10	10.0	.4	.8
Modified Langerman	5	20.0	.2	.4
	10	30.0	.4	.8
Shekel's Foxholes	5	24.0	-.4	0.0
	10	24.0	-.5	0.0
Epistatic Michalewicz	5	4.0	.5	.3
	10	1.2	.5	.2
Tchebychev Polynomial	9	10.0	.5	1.0
	17	10.0	.5	1.0
The Bump	10	2.0	.5	0.0
	20	2.0	.5	0.0
Individual Crafting Effort	$\log_2$ ln $\log_{10}$	2.807 1.946 .8451	2.118 1.468 .6376	2.324 1.611 .6998
Total Crafting Effort	$\log_2$ ln $\log_{10}$	>	7.249 5.025 2.183	<

times the dimension of the function under investigation. The DE settings used to achieve the results reported in Table 1 are presented in Table 2, along with the resulting crafting measure that they incur. For comparison, the crafting effort has been computed using base 2, base 10, and natural logarithms.

VI. Conclusions

Convergence to a global minimum is the most important demand that can be made of a numerical optimizer, although the speed with which it does so is also an important consideration. Even though it was unable to resolve the minimum of the 10-D Odd Square function within the prescribed number of evaluations, DE found an optimal vector in every other case. For the Rosenbrock, Shekel, Tchebychev and Bump functions, DE located the optimal vector in all twenty trials, but DE was unable to successfully optimize the remaining functions for every trial within the allowed number of evaluations. A moderate increase in the number of allowed evaluations, however, would have been sufficient to enable DE to converge with regularity across the test bed except for all but the 10-D Odd Square function. Not only would a small increase in the maximum allowed number of evaluations significantly improve the regularity with which DE found test bed optima, it would also reduce the crafting effort needed to find them. Thus, DE is, perhaps, more robust than the results presented here indicate, due primarily to the contest's emphasis on speed of convergence.

Although a final judgment regarding its relative value as a global optimizer must await a comparison with its competitors, DE appears to have performed well on the 2nd ICEO test bed, especially when one considers how simple the algorithm is.

Acknowledgment

The author would like to thank Dr. Rainer Storn for his considerable input during the development of DE, and for his valuable assistance in bringing DE to a wider audience.

References

[1] H. Bersini, M. Dorigo, L. Gambarella, S. Langerman and G. Seront, "Results of the First International Contest on Evolutionary Optimisation (1st ICEO)", *Proc. of the 1996 IEEE International Conference on Evolutionary Computation*, Nagoya, Japan, 1996, ISBN:0-7803-2902-3.

[2] R.Storn and K. Price, "Minimizing the Real Functions of the ICEC '96 Contest by Differential Evolution", *Proc. of the 1996 IEEE International Conference on Evolutionary Computation*, Nagoya, Japan, 1996, ISBN:0-7803-2902-3.

[3] K. Price, "Genetic Annealing", *Dr. Dobb's Journal*, Oct. 1994, Miller Freeman, San Mateo, Ca., 1994, pp. 127-132.

[4] T. Back, G. Rudolph and H.-P. Schwefel, "Evolutionary programming and evolutionary strategies: similarities and differences", *Proc. of the Second Annual Conference on Evolutionary Programming*, D.B. Fogel and W. Atmar, (eds.), Evolutionary Programming Society, La Jolla, Ca, 1993, pp. 11-22.

[5] K. Price, "Differential Evolution - A Fast and Simple Numerical Optimizer", *Proc. of 1996 North American Fuzzy Information Processing Society*, Berkeley, Ca., 1996, ISBN:0-7803-3225-3.