Minimizing the real functions of the ICEC'96 contest by Differential Evolution

Rainer Storn

Siemens AG, ZFE T SN2, Otto-Hahn Ring 6, D-81739 Muenchen, Germany, currently on leave at ICSI, 1947 Center Street, Berkeley, CA 94704, storn@icsi.berkeley.edu

Kenneth Price

836 Owl Circle, Vacaville, CA 95687, kprice@solano.community.net

Abstract

Differential Evolution (DE) has recently proven to be an efficient method for optimizing real-valued multi-modal objective functions. Besides its good convergence properties and suitability for parallelization, DE's main assets are its conceptual simplicity and ease of use. This paper describes two variants of DE and summarizes their performance on the real test functions of the ICEC'96 contest.

Introduction

Differential Evolution (DE) [1], [2] has proven to be a promising candidate for optimizing real-valued multi-modal objective functions. Besides its good convergence properties DE is very simple to understand and to implement. DE is also particularly easy to work with, having only a few control variables which remain fixed throughout the entire optimization procedure.

DE is a parallel direct search method which utilizes NP D-dimensional parameter vectors:

$$\underline{x}_{i,G},\ i = 0, 1, 2, \ldots, NP-1, \qquad (1)$$

as a population for each generation G, i.e. for each iteration of the optimization. NP doesn't change during the minimization process. The initial population is chosen randomly and should try to cover the entire parameter space uniformly. As a rule, we will assume a uniform probability distribution for all random decisions unless otherwise stated. The crucial idea behind DE is a scheme for generating trial parameter vectors. Basically, DE generates new parameter vectors by adding the weighted difference between two population vectors to a third vector. If the resulting vector yields a lower objective function value than a predetermined population member, the newly generated vector will replace the vector with which it was compared in the following generation; otherwise, the old vector is retained. This basic principle, however, is extended when it comes to the practical variants of DE. For example an existing vector can be perturbed by adding more than one weighted difference vector to it. In most cases, it is also worthwhile to mix the parameters of the old vector with those of the perturbed one. The performance of the resulting vector is then compared to that of the old vector. We will describe two variants of DE which have proven to be useful.

Variant DE/rand/1

For each vector $\underline{x}_{i,G}$, i = 0,1,2,...,NP-1, a perturbed vector $\underline{v}_{i,G+1}$ is generated according to

$$\underline{v}_{i,G+1} = \underline{x}_{r_1,G} + F \cdot (\underline{x}_{r_2,G} - \underline{x}_{r3,G}) \qquad (2)$$

with $r_1, r_2, r_3 \in [0, NP-1]$, integer and mutually different, and F > 0.

The integers r_1, r_2 and r_3 are chosen randomly from the interval [1, NP] and are different from the running index i. F is a real and constant factor $\in$ [0, 2] which controls the amplification of the differential variation $(\underline{x}_{r_2,G} - \underline{x}_{r_3,G})$. Note that the vector $\underline{x}_{r_1,G}$ which is perturbed to yield $\underline{v}_{i,G+1}$ has no relation to $\underline{x}_{i,G}$ but is a randomly chosen population member. Fig. 1 is a two

dimensional example that illustrates the different vectors which play a part in the vector-generation scheme. The notation: DE/rand/1 specifies that the vector to be perturbed is randomly chosen and that the perturbation consists of one weighted difference vector.

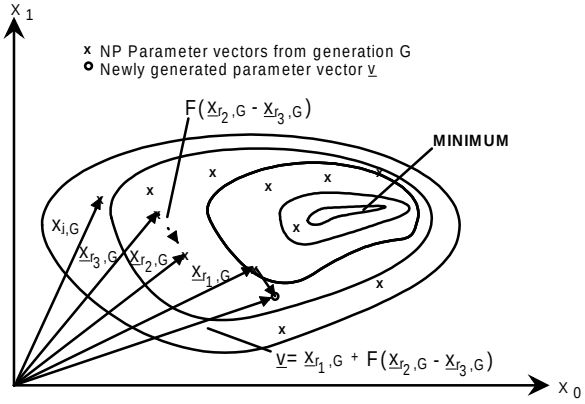

Fig.1: An example of a two-dimensional objective function showing its contour lines and the process for generating $\underline{v}$ in scheme DE/rand/1.

In order to increase the diversity of the new parameter vectors, crossover is introduced. To this end, the vector:

$$\underline{u}_{i,G+1} = (u_{0i,G+1}, u_{1i,G+1}, \ldots, u_{(D-1)i,G+1}) \qquad (3)$$

with

$$u_{ji,G+1} = \begin{cases} v_{ji,G+1} & for \;\; j = \langle n \rangle_D, \langle n+1 \rangle_D, \ldots, \langle n+L-1 \rangle_D \\ x_{ji,G} & for \;\; all \;\; other \;\; j \in [0, D-1] \end{cases} \qquad (4)$$

formed. The acute brackets $\langle \; \rangle_D$ denote the modulo function with modulus D. The starting index, n, in (4) is a randomly chosen integer from the interval [0,D-1]. The integer L, which denotes the number of parameters that are going to be exchanged, is drawn from the interval [1, D]. The algorithm which determines L works according to the following lines of pseudo code where rand() is supposed to generate a random number $\in$ [0,1):

```
L = 0;
do {
    L = L + 1;
}while(rand()< CR)  AND  (L < D));
```

Hence the probability Pr(L=ν) = $(CR)^{\nu-1}$, ν > 0. CR $\in$ [0,1] is the crossover probability and constitutes a control variable in the design process. The random decisions for both n and L are made anew for each newly generated vector $\underline{u}_{i,G+1}$. Fig. 2 provides a pictorial representation of DE's crossover mechanism.

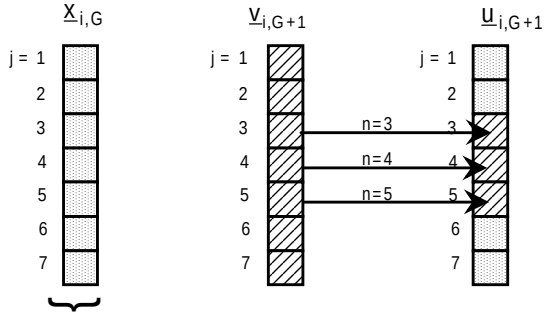

Fig. 2: Illustration of the crossover process for D=7, n=2 and L=3.

To decide whether or not it should become a member of generation G+1, the new vector $\underline{u}_{i,G+1}$ is compared to $\underline{x}_{i,G}$. If vector $\underline{u}_{i,G+1}$ yields a smaller objective function value than $\underline{x}_{i,G}$, then $\underline{x}_{i,G+1}$ is replaced by $\underline{u}_{i,G+1}$; otherwise, the old value $\underline{x}_{i,G}$ is retained.

The following pseudo-code illustrates how DE basically works. Comments are preceded by // like in C++. A two-dimensional array, e.g. pop, is indicated by pop[][]. A vector, e.g. temp, is denoted by temp[]:

```
//---Initialize vector population--------
initialize(pop_new[][]);
initialize(best[]);//---best vector so far
initialize(value_best);//---best value so far
```

```
//---do for itermax iterations------
for j=1 to itermax
{//---current vector population in pop[][]
pop[][] = pop_new[][];
//---scan through all NP vectors----
for i=0 to NP-1
{
   //---pick three randomly chosen---
   //---population members-----------
   r1 = random integer ex [0,NP-1]
        but not i;
   r2 = random integer ex [0,NP-1]
        but not i and not r1;
   r3 = random integer ex [0,NP-1]
        but not i, not r1 and not r2;
   //---save population member no. i----
   //---in temporary vector-------------
   for j=0 to D-1
   {
      temp[i] = pop[i][j];
   // parameter j from population member i
   }
   //---do crossover--------------------
    n = random integer ex [0, D-1];
    L = 0;
    do {
      //---equ. (2)----------------------
      temp[n] = pop[r1][n] +
      F*(pop[r2][n] - pop[r3][n]);
      n = (n + 1)modD; //--wrap around
      L = L + 1;
    }while(rand()< CR)  AND  (L < D));
   //---Evaluate new vector temp---------
   value_new = evaluate(temp[]);
   value_old = evaluate(pop[i]);
   if (value_new < value_old)
   { //---if new vector better-----------
      pop_new[i][] = temp[]; //it survives
      if (value_new < value_best)
      { //---all time best ?-------------
         value_best = value_new;
         best[] = temp[]; //save best member
      }
   }
   else //---retain old vector------------
   {
      pop_new[i][] = pop_old[i][];
   }

}//---End for i=0 to NP-1

}//---End for j=1 to itermax
```

Scheme DE/best/2

Basically, scheme DE/best/2 works the same way as DE/rand/1 except that it generates the vector $\underline{v}$ according to:

$$\underline{v} = \underline{x}_{best,G} + F \cdot (\underline{x}_{r_1,G} + \underline{x}_{r_2,G} - \underline{x}_{r_3,G} - \underline{x}_{r_4,G}). \quad (5)$$

This time, the vector to be perturbed is the best performing vector of the current generation and two weighted difference vectors are used for the perturbation. The construction of $\underline{u}$ from $\underline{v}$ and $\underline{X}_{i,G}$ as well as the criterion for its acceptance are identical to DE/rand/1. It is simple to change the pseudo code of DE/rand/1 so that it works according to the DE/best/2 strategy: firstly, a new random variable r4 has to be introduced and secondly, the evaluation of temp[] in the do-loop has to be changed so that it computes eq. (5) instead of eq. (2).

Conclusion

Two variants of Differential Evolution (DE) have been used to solve the ten test problems of the ICEC'96 contest. It turned out that both approaches could solve all ten problems without failure (see table I) which is interpreted as further evidence that DE is an efficient function minimizer. In addition to its good convergence properties, DE exhibits structural and operational simplicity and is well suited for parallelization.

References

[1] Storn, R. and Price, K., Differential Evolution - a simple and efficient adaptive scheme for global optimization over continuous spaces, Technical Report TR-95-012, ICSI, http://http.icsi.berkeley.edu/~storn/litera.html

[2] Storn, R., Differential Evolution Design of an IIR-Filter, ICEC'96.

	DE/rand/1					DE/best/2				
Problem	NP	F	CR	ENES	RT	NP	F	CR	ENES	RT
1 (5D)	8	0.5	0.	736	4.67	5	0.5	0.1	463	7.40
1 (10D)	10	0.5	0.4	1892	4.88	5	0.45	0.	1187	6.66
2 (5D)	20	0.5	0.1	5765	1.79	15	0.5	0.2	5157	1.79
2 (10D)	25	0.5	0.2	13508	1.77	25	0.5	0.2	16228	1.67
3 (5D)	200	0.75	0.	76210	0.80	200	0.99	0.	67380	0.81
3 (10D)	500	1.2	0.	744250	0.66	200	1.5	0.	203350	0.60
4 (5D)	20	0.5	0.	1877	1.11	25	0.6	0.	2551	1.12
4 (10D)	30	0.9	0.	10083	0.68	40	0.75	0.	18158	0.67
5 (5D)	15	0.8	0.4	5308	1.35	20	0.7	0.5	4814	1.38
5 (10D)	60	0.6	0.8	44733	1.46	120	0.7	0.7	174006	1.28

Table I: Results of minimizing of the ICEC'96 functions with DE.