Development and validation of constraints handling in a Differential Evolution optimizer

Mihai-Vladut HOTHAZIE*,[1], Georgiana ICHIM[1], Mihai-Victor PRICOP[2]

*Corresponding author
[1]University "POLITEHNICA" of Bucharest, Faculty of Aerospace Engineering,
str. Polizu 1, Bucharest, Romania, 011061,
vlad.hothazie@gmail.com*, georgianaichim.gi@gmail.com
[2]INCAS – National Institute for Aerospace Research "Elie Carafoli",
B-dul Iuliu Maniu 220, Bucharest 061126, Romania,
pricop.victor@incas.ro

DOI: 10.13111/2066-8201.2020.12.1.6

Received: 06 January 2020/ Accepted: 05 February 2020/ Published: March 2020

The 38th "Caius Iacob" Conference on Fluid Mechanics and its Technical Applications
7 - 8 November, 2019, Bucharest, Romania, (held at INCAS, B-dul Iuliu Maniu 220, sector 6)
Section 4. Mathematical Modeling

Abstract: *Research work requires independent, portable optimization tools for many applications, most often for problems where derivability of objective functions is not satisfied. Differential evolution optimization represents an alternative to the more complex, encryption based genetic algorithms. Various packages are available as freeware, but they lack constraints handling, while constrained optimizations packages are commercially available. However, the literature devoted to constraints treatment is significant and the current work is devoted to the implementation of such an optimizer, to be applied in low-fidelity optimization processes. The parameter free penalty scheme is adopted for implementation, and the code is validated against the CEC2006 benchmark test problems and compared with the genetic algorithm in MATLAB. Our paper underlines the implementation of constrained differential evolution by varying two parameters, a predefined parameter for feasibility and the scaling factor, to ensure the convergence of the solution.*

Key Words: *evolutionary algorithm, differential evolution, constraints, optimization*

1. INTRODUCTION

Nowadays, evolutionary algorithms (EAs) are widely used to solve optimization problems, non-linear, non-convex, multi-modal and non-differentiable [1] functions in the continuous parameter space. In its original form, EA is not compatible to perform operations with constraint problems. Thus, a multitude of methods have been developed to search for the feasible region of the fitness function. The most popular approach is the use of penalty functions [2]. Even so, the drawback of the penalty function is that it must be carefully chosen, depending on the constraint problem, thus, losing the generality.

Differential Evolution algorithm (DEC) was introduced by Storn and Price [3] and it is primarily used for constraint optimization problems due to its simplicity and performance.

DE consist of a uniform population consisting of a random set of candidate solutions, known as target vectors, from the given search space. Compared to a basic evolutionary algorithm, DE implies a secondary parent, namely a mutation vector which scales the difference of two random members of the current population and added to another member. The scaling factor "F" scales the pair of random vectors to obtain the donor/mutant vector. Next, the crossover between a target vector and the mutant vector is done to form an offspring vector. The crossover parameter "CR" defines the influence of the parent in the next generation [4]. In the end a selection is undergone to determine whether the target vector or the offspring survives for the next generation. Therefore, if the offspring yields an equal or lower value of the objective function, it becomes the new parent in the next generation. With every iteration, the population tends to adapt to the natural scale of the feasible landscape, converging to the optimal solution.

The purpose of this paper is to implement the Differential Evolution algorithm in a standard programming language (FORTRAN 9x), enforcing our own contribution so that it is robust, compact and reliable, with a maximal computational speed. The following approach consists of an initial population of at least 50% feasible target vectors and for every iteration, the scaling factor F is a uniformly distributed random number lying between 0 and 2. To accurately demonstrate the performance of the algorithm, a set of standard benchmark functions with nonlinear constraints are minimized.

2. METHODS

Constrained systems brought a subtle challenge in finding the optimum. These constraints are coming from the physical modeling of various systems. In most cases the optimal value changes, calling for different handling techniques on finding the minimum. A general constrained optimization problem with n parameters to be optimized is usually written as a nonlinear programming problem of the following form:

Minimize $f(x)$ subjected to [5]

$$g_k(x) \leq 0, k = 1, \dots K, \tag{1}$$

$$h_l(x) = 0, l = 1, \dots L, L < n, \tag{2}$$

$$l^k \leq x^j \leq u^j, j = 1, \dots n. \tag{3}$$

Where $g_k(x)$ define the inequality constraints and $h_l(x)$, the equality constraints. The lower bounds and the upper bounds within the variables lie are $[l^j, u^j]$. Thus, the feasible region is given by:

$$\Omega = \{x = [x^1, x^2, \dots x^n] \in \mathbb{R}^n | g_k(x) \leq 0, h_l(x) = 0, l^j \leq x^j \leq u^j, \forall j\} \tag{4}$$

Regarding our perspective on the constrained optimization, when we deal with equality constraints they are written as follows:

$$g_l(x) = h_l(x) - \varepsilon \leq 0 \tag{5}$$

$$g_{l+1}(x) = -h_l(x) - \varepsilon \leq 0 \tag{6}$$

Having ε as a small positive value that is automatically changed, depending on the objective function. The constraint handling technique used in the approach of the evolutionary algorithm is based on preservation of the feasibility of solutions. Namely we restrict the search

process to the boundary of the feasible space. The major limitation is finding a feasible starting point and consequently a feasible population. Different constraint problems may have a very small search design space thus, the process of localizing this space becomes difficult regarding time and processing.

2.1 The parameter free penalty-scheme [5]

Implemented by Deb in 2000, the parameter-free penalty scheme (PFP) is similar to the superiority feasible points only that it lacks the penalty coefficient. In this scheme, we use a modified fitness function in order to ensure that the feasible points will always have a better fitness function than unfeasible ones.

The modified fitness function for this scheme is obtain as follows:

$$\hat{f}(x_{i,g}) = f(x_{i,g}) + G(x_{i,g}) + \theta_g(x_{i,g}), x_{i,g} \in s_g \tag{7}$$

where $f(x_{i,g})$ is the original fitness function, $G(x_{i,g})$ is the constraint violation function and $\theta_g(x_{i,g})$ is an additional penalty term which guarantees that the infeasible points will always have a worse modified fitness function. The additional penalty term has the next possible cases:

$$\Theta_g(x_{i,g}) = \begin{Bmatrix} 0 & if & x_{i,g} \in \Omega \\ -f(x_{i,g}) & if & s_g \in \Omega = \emptyset \\ -f(x_{i,g}) + max_{y \in s_g \cap \Omega f(y)} & if & s_g \in \Omega = \emptyset, \quad x_{i,g} \notin \Omega \end{Bmatrix} \tag{8}$$

The main use of this penalty term is to direct the search process to a feasible region but still maintaining a random factor in order to keep the highest chances of finding the global minimum and not a local one. This penalty term can have different values depending on the number of feasible points in our population.

The first case deals with an ideal population where all the points are feasible. Thus, the penalty term is equal to 0 because there is no need to penalize points that are in the feasible region. Therefore, the modified fitness function will be equal only to the sum of the constraint violation function and the original fitness function.

$$\hat{f}(x_{i,g}) = f(x_{i,g})$$
$$G(x_{i,g}) = 0$$
$$\theta(x_{i,g}) = 0$$

In the second instance, if the population consists only of infeasible points, the penalty function will be equal to the negative of the original fitness function. This procedure enables an improvement in the calculation time because the modified fitness function will only be equal to the constraint violation function. As such, there is no need to evaluate the original fitness function, and in many cases, this is a major improvement.

$$\hat{f}(x_{i,g}) = f(x_{i,g})$$
$$G(x_{i,g}) > 0$$
$$\theta(x_{i,g}) = -f(x_{i,g})$$

In the last situation, if there are both feasible and unfeasible points in the population, the penalty term for an unfeasible point will take the sum of the maximum value of the original fitness function from all the feasible points minus the original fitness function for the

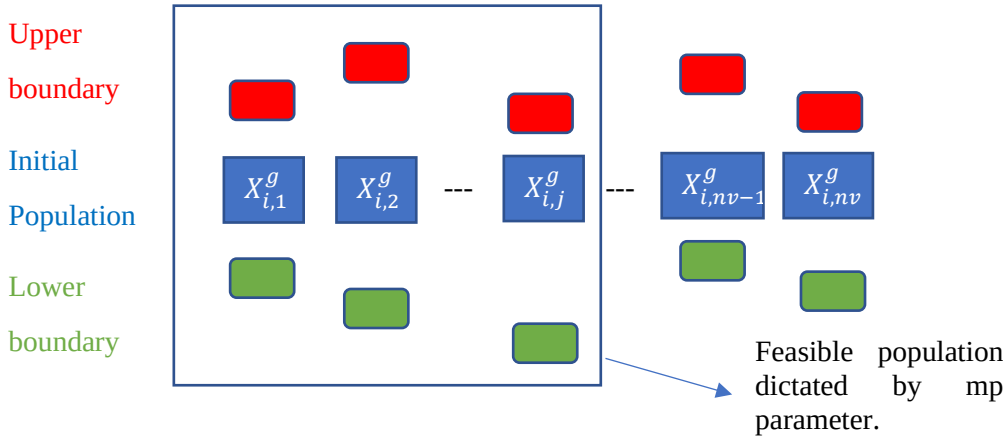
respective unfeasible point. This ensures that every unfeasible point will always have a worse modified fitness function then the feasible ones. So, the search will be directed to the feasible ones.

$$\hat{f}(x_{i,g}) = G(x_{i,g}) + max_{fp}(f(x_{i,g}))$$
$$G(x_{i,g}) \geq 0$$
$$\theta(x_{i,g}) = -f(x_{i,g}) + max_{fp}(f(x_{i,g}))$$

The resulting fitness function is the most appropriate choice for solving problems with constraints and also encourages a relatively modified basic structure of the usual algorithm. A standard differential algorithm is used but slightly different to obtain the best solution in a time-efficient way. Four main stages are necessarily required at every iteration, presented as follows:

1. Initial population

In order to start the process, we generate a random population. To ensure that our code has a higher convergence rate, we impose that a minimum percent of this startup randomly selected individuals is feasible and higher than a parameter ***mp*** that can be modified in the program.



2. Mutation population

To obtain mutated individuals, the algorithm uses the difference between two randomly selected points from the parent population as a root for a third individual, also randomly chosen, in order to spawn a new element. This method proves to be a good approach because if we repeat the process for a number of iterations the point converges to a feasible region.

This method can have different forms. The one used in our code is:

$$\hat{X}_i^{g+1} = X_{r1}^g + F \cdot \left(X_{r2}^g - X_{r2}^g\right) \tag{9}$$

where r_1, r_2, r_3 are randomly generated integers within the total number of the elements from the population. F represents the scaling factor and can be used as a fixed parameter or can be initiate randomly for every point.

However, it can be easily changed to other alternatives. For example:

$$\hat{X}_i^{g+1} = X_{r1}^g + F^2 \cdot \left(X_{r2}^g - X_{r2}^g\right) \tag{10}$$

$$\hat{X}_i^{g+1} = X_{r1}^g + F \cdot \left(X_{r2}^g - X_{r2}^g\right) + F^2 \cdot \left(X_{r2}^g - X_{r2}^g\right) \tag{11}$$

$$\hat{X}_i^{g+1} = X_{r1}^g + F_1 \cdot \left(X_{r2}^g - X_{r2}^g\right) + F_2 \cdot \left(X_{r2}^g - X_{r2}^g\right) \quad (12)$$

In this phase, another parameter (NMP) is added. It corresponds to the minimum number of feasible elements necessary to complete the mutation population. The process is repeated until the NMP satisfies a percentage added by the user.

3. Cross-over population

The cross-over is the next phase of the algorithm. For each mutated vector, a trial one is generated by using the following rule:

a. A cross-over parameter $cr \in (0,1)$ is considered and two random numbers are generated: $R_i \in rand(0,1)$, $I_i \in \overline{1,np}$.
b. Every individual from the cross-over vector is selected from the mutated vector if $R_i < cr$ or $I_i = j$. Otherwise, it is picked from the parent vector. This ensures that at least one element from the parent population is taken into account in the newly created vector.

$$Y_{i,g} = \begin{cases} \hat{X}_{i,j} & if \;\; R_i < cr \;\; or \;\; I_i = j \\ X_{i,j} & otherwise \end{cases} \quad (13)$$

4. **Acceptance phase**

A selection criterion is applied to create the new parent population for the next generation. The acceptance operator implies a one-to-one competition between two points: one from the current parent population and one from the cross-over population.

$$X_i^{g+1} = \begin{cases} \hat{X}_{i,j} & if \;\; \hat{f}(X_i^g) < \hat{f}(Y_i^g) \\ Y_i^g & otherwise \end{cases} \quad (14)$$

We now present the DEC algorithm below:

1. Set control parameters: $\boldsymbol{mp}, \boldsymbol{cr}, \boldsymbol{f}, \boldsymbol{eps}$.
2. **IF** condition for $\boldsymbol{mp}$ not met:
   a. Generate initial population
   b. Evaluate modified objective function $\hat{f}(x_{i,g})$, constraint violation function $G(x_{i,g})$ and the penalty function $\theta(x_{i,g})$ for each point in the population.
3. Determine the penalty function $\theta(x_{i,g})$ and compute the modified fitness function $\hat{f}(x_{i,g})$ once again for the initial population.
4. **IF** stopping criteria not met:
   a. Generate mutated population;
   b. Generate cross-over population;
   c. Evaluate $G(x_{i,g})$, $\hat{f}(x_{i,g})$, $\theta(x_{i,g})$;
   d. Update parent population.

3. RESULTS

In the extent of running our algorithm for each minimization problem in particular, the following results presented in *Table 1* have been obtained. The size of population and number of generations are chosen depending on the convergence of each problem. The number of generations for each problem ran with differential evolution is represented in the table below with "nGen". Denoted with "%" is a predefined parameter which implies what percentage of

the initial population must be feasible so that the algorithm continues running. The crossover parameter "CR" has a fixed value and ε is an adjustable parameter for merging the equality constraints into inequalities. Our obtained value of the minimize function is represented in the table as $f(\vec{x}^*)$. The last column of the table, "fval", has the values obtained with the genetic algorithm.

To make a fair comparison between the results, a cross-over parameter is set. In our tests, we observed that this parameter has a high accuracy rate if is above 0.7. The number of generations for each problem is chosen in order to converge as fast as possible. The population size is desired to be as low as possible, but maintaining the best result.

Table 1 - Results

Prob.	Pop.	nGen	%	CR	ε	$f(x)$	fval
g01	10^2	10^4	0.8	0.9	-	-0.1500000000E+02	-0.1500112511E+02
g02	10^2	10^8	0.8	0.9	-	-0.8029606447E+00	**-0.1809705412E+00**
g03	10^2	10^6	0.8	0.9	10^{-4}	-0.1000044328E+01	-0.9979858426E+00
g04	10^2	10^4	0.5	0.9	10^{-2}	-0.3060118322E+05	-0.3007533718E+05
g05	No feasible point found						0.54184790878E+04
g06	10^2	5.e^6	0.5	0.9	-	-0.6932086083E+04	-0.6963708061E+04
g07	10^2	2.e4	0.8	0.9	-	0.2441341031E+02	0.2586227704E+02
g08	10^2	1.e3	0.8	0.9	-	-0.9582504142E-01	-0.9582050424E-01
g09	10^3	1.e4	0.8	0.9	-	0.6812588890E+03	0.6819709217E+03
g10	No feasible point found						
g11	10^2	1.e3	0.8	0.9	10^{-3}	0.7496893234E+00	0.7499988873E+00
g13	10^3	5.e4	0.5	0.9	-	0.5484121102E-01	**0.1490996960E+00**
g14							0.4719957105E+00
g15	10^2	10^3	0.8	0.9	10^{-1}	0.9582137398E+03	0.96171521921E+03
g18							-0.8500959489E-03
g24	10^2	10^4	0.8	0.9	-	-0.5508013272E+01	-0.5509003070E+01
P04	10^3	12.e4	0.8	0.9	10^{-2}	-0.4532419854E+01	-0.4453184391E+01
P06	10^2	10^4	0.8	0.9	10^{-1}	-0.1286788746E+02	-0.1264548378E+02
P08	10^3	3.e3	0.8	0.9	10^{-5}	-0.1673888371E+02	-0.1657674053E+02

The optimum solution for each problem is presented in *Table 2.* All variables are compared with a high order of accuracy to the specified constraints for each problem to make a comparison as clean as possible.

Table 2 - Solutions

g01	0.4709424817E+00	0.8694350187E+00	-0.1652805832E+01
0.1000000000E+01	0.4549283240E+00	**g07**	0.1519419318E+01
0.1000000000E+01	0.4731322305E+00	0.2155801854E+01	0.1930927354E+01
0.1000000000E+01	0.4331673978E+00	0.2413759969E+01	0.6125975779E+00
0.1000000000E+01	0.4566534626E+00	0.8742887898E+01	0.9773569805E+00
0.1000000000E+01	0.4490200205E+00	0.5036086968E+01	**g15**
0.1000000000E+01	0.4397910488E+00	0.9865799789E+00	0.5331076897E+01
0.1000000000E+01	0.4193101507E+00	0.1400319589E+01	0.3191077212E-01
0.1000000000E+01	**g03**	0.1287114899E+01	0.1839882632E+01
0.1000000000E+01	0.3159080639E+00	0.9796314515E+01	**g24**

0.3000000000E+01	0.3167447079E+00	0.8296434608E+01	0.2329520197E+01
0.3000000000E+01	0.3165286793E+00	0.8532309349E+01	0.3178493074E+01
0.3000000000E+01	0.3161828299E+00	**g08**	**P04**
0.1000000000E+01	0.3161392258E+00	0.1227971352E+01	0.1336666666E+01
g02	0.3162191244E+00	0.4245373367E+01	0.4000000000E+01
0.3151573513E+01	0.3160117527E+00	**g09**	0.6305358251E-10
0.3118072951E+01	0.3160833764E+00	0.2149871648E+01	0.1189899211E-16
0.3086665232E+01	0.3163388956E+00	0.1951091213E+01	**P06**
0.3055059035E+01	0.3161359041E+00	-0.3406304372E+00	0.1639924104E+00
0.3021179622E+01	**g04**	0.4394110482E+01	0.1889573051E+01
0.2980726117E+01	0.7800000000E+02	-0.5257916209E+00	0.3882797751E+01
0.2944331331E+01	0.3300789591E+02	0.1118017674E+01	0.4671558308E+00
0.2920353116E+01	0.3021031943E+02	0.1517986013E+01	0.3456871167E-01
0.5095040982E+00	0.4390435446E+02	**g11**	0.1967281535E+01
0.5069184137E+00	0.3669881882E+02	0.6908307223E+00	**P08**
0.4987658001E+00	**g06**	0.4780400049E+00	0.7169765266E+00
0.4887158913E+00	0.1410729830E+02	**g13**	0.1471494488E+01

4. CONCLUSIONS

The results support our initial hypothesis, that the minimum of the function is found with better accuracy using the parameter-free penalty scheme. This shows that the unfeasible solutions are penalized so that they do not interfere with the feasible ones.

The results are close, if not the same, with the ones given in CEC2016 which emphasize the performance of the algorithm. In addition to this comparison, a set of results obtained with the genetic algorithm from MATLAB Global Optimization Toolbox is added to the list. A difference is noted between our differential evolution algorithm and the GA, namely, the number of generations and the size of the population are fixed in GA. If changed, the runtime is larger and there are chances for the problem not to converge to the optimal solutions and fall to an unfeasible one as there is no scheme used. The solution obtained with no parameter changed in the standard GA is worse than the one obtained with DE.

In the case of problem g05 and g10, the algorithm did not converge because there was no initial population with a minimum of 50% feasible elements. The values obtained with the genetic algorithm shown in *Table 1* in bold are a clear example of how the solution fall into a local minimum, thus unfeasible.

Nowadays, the main challenging problem is to find a robust and intelligent computational scheme that can be easily applied to solve a problem that cannot be determined analytically or is too complicated or time-consuming.

The algorithm should respect a set of rules, so it can be easily understood by engineers and efficiently applied for the specific problem.

1. The scheme should not require an initial feasible population inserted by the user in order to converge.
2. It should not be too sensitive if one or more parameter is changed.

Taking everything into account, our program achieved performance comparable to or better than the MATLAB genetic algorithm. In addition, although there are some variants of this algorithm on the market, they aren't free to use. Therefore, for our institute, our code will be accessible to anyone in order to facilitate optimized results for specific problems. Furthermore, the basic structure was designed to be modular so it can be easily debugged and modified for their specific needs.

REFERENCES

[1] S. Das, S. Subhra Mullick, P. N. Suganthan, *Recent advances in differential evolution - An updated survey*, Swarm and Evolutionary Computation, **27**: 1–30, Elsevier, 2016.

[2] R. Storn, K. Price, Differential Evolution – A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces, *Journal of Global Optimization*, **11**: 341–359, Kluwer Academic Publishers, 1997.

[3] E. Mezura-Montes, J. Velazquez-Reyes and C. A. Coello Coello, *Modified Differential Evolution for Constrained Optimization*, *Evolutionary*, Computation Group (EVOCINV) at CINVESTAV-IPN, Computer Science Section, Electrical Engineering Department, 2508 Col. San Pedro Zacatenco M´exico D.F. 07300, MEXICO, 2006.

[4] S. Dominguez-Isidro, *Memetic Differential Evolution for Constrained Numerical Optimization Problems*, Doctoral Thesis, University of Veracruz, 2017.

[5] Z. Kajee-Bagdadi, *Differential Evolution Algorithms for Constrained Global Optimization*, Master Thesis, Faculty of Science, University of the Witwaterstrand, Johannesburg, 2007.