A Constraint Handling Approach for the Differential Evolution Algorithm

Jouni Lampinen
Lappeenranta University of Technology
Laboratory of Information Processing
P.O.Box 20, FIN-53851 Lappeenranta, Finland

Abstract - An extension for the Differential Evolution algorithm is proposed for handling nonlinear constraint functions. In comparison with the original algorithm, only the replacement criterion was modified for handling the constraints. In this article the proposed method is described and demonstrated by solving a suite of ten well-known test problems.

I. INTRODUCTION

Evolutionary algorithms have received a lot of attention regarding their potential for solving effectively a wide range of difficult optimization problems, including global optimization problems subject to multiple nonlinear constraints. A wide variety of evolutionary algorithm based approaches have been proposed for solving multi-constrained nonlinear problems [11]. However, handling multiple nonlinear constraint functions has been found rather difficult, and no totally satisfying method appears to be available, when considering all aspects involved.

In [11] Michalewicz and Schoenauer have surveyed evolutionary algorithms for constrained parameter optimization problems. They have classified the constraint handling methods applied with evolutionary algorithms into four categories:

- methods based on preserving feasibility of solutions,
- methods based on penalty functions,
- methods which make a clear distinction between feasible and infeasible solutions, and
- other hybrids.

In general, the first two categories are undoubtedly the most widely applied with all types of nonlinear optimization algorithms, while the remaining two categories include a wide variety of less frequently applied approaches.

The characteristic problems with the most frequently applied approaches are rather well known. Methods preserving feasibility of solutions often require providing a feasible starting point for the search process. Unfortunately, a feasible starting point is often impossible to provide, since the feasible region may be only a small subset of the whole search space. Finding a feasible starting point randomly is often hopeless, or it takes much more time than the actual optimum seeking process, after first finding a feasible starting point. Preferably, the optimum-seeking algorithm should be capable of finding feasible solutions itself.

Another disadvantage is that maintaining the feasibility of solutions often result in sticking into a suboptimal solution in cases, where the feasible solutions form a number of isolated islands around the search space due to multiple nonlinear constraints. This situation is typical for practical applications.

These problems can often be avoided by using penalty function approaches. However, typically penalty function methods require setting additional search parameters, like various penalty parameters or weighting factors for individual constraint functions. It is well known that the selected values for these search parameters affect on the convergence performance as well as any other algorithm specific search parameter. Thus selecting suitable values can be laborious, or in cases of many practical applications, even too difficult.

In particular with the evolutionary algorithms, *underpenalization* of the infeasible solutions (e.g. applying too low weights for each constraint function) typically result in slow convergence towards the feasible solutions, or in worst case, no feasible solution will be found at all. *Overpenalization* typically results in a rapid convergence to a feasible solution. Unfortunately, often a premature convergence to suboptimal solution occurs simultaneously. Due to too high convergence rate, the search space will be explored poorly during the early stages of the search process. If the feasible solutions form multiple disjointed regions around the search space, the population tends to converge to the first found island of feasible solutions, without searching the others sufficiently.

Especially, if the number of constraint functions is relatively high, say >10, finding the best, or even acceptable penalty parameter values is actually a difficult optimization problem itself. Typically the user does not have enough problem specific information available for selecting these settings adequately *a priori*. So, the only alternative is to set the penalty parameters by applying an educated guess at first, and then attempting to refine the settings by the trial-and-error method. This process is both laborious and time consuming – it is rare to solve a nonlinearly constrained problem with a single run of the optimization routine using the originally guessed penalty parameter settings.

Most penalty function methods do not require a feasible starting point, and they are capable of providing a nearest feasible solution if the feasible set appears to be empty. This is an important character, since in real-life problems it is often impossible to judge *a priori*, whether there are any feasible solutions at all for the problem. After having the nearest feasible solution and the constraint function values at hand, it is often easy to judge which constraints are violated, why they are violated, and perhaps even reconsider the problem statements to solve the problem with the empty feasible set.

One of the most promising novel evolutionary algorithms is the Differential Evolution (DE) algorithm for solving global optimization problems with continuous parameters. The DE was first introduced a few years ago by Price and Storn [12, 13]. Subsequently, Lampinen and Zelinka [8] have proposed an extension of DE algorithm for mixed-discrete optimization including the required methods for handling also discrete and integer valued parameters.

0-7803-7282-4/02/$10.00 ©2002 IEEE

Based on the literature referring to usage of DE for solving constrained problems [7], mainly various penalty function methods have been applied for handling constraint functions in connection with DE. The objective of this investigation was finding an alternative constraint handling method for DE, preferably keeping the advantages of the penalty function approach, but simultaneously gets rid of laborious and often troublesome setting of the penalty parameter values. Thus the target was developing an effective constraint handling method, which does not require user to set any additional search parameters in comparison with the original DE, and which does not require any extra actions by the user, except providing the constraint functions when programming the evaluation function routine.

II. DIFFERENTIAL EVOLUTION ALGORITHM

The DE algorithm can be classified as *a floating-point encoded evolutionary optimization algorithm.* At present, there are several variants of DE [12]. The particular version used throughout this investigation was the *DE/rand/1/bin* scheme, which is briefly described in the following.

A. The Problem

Generally, a multi-constrained nonlinear optimization problem can be expressed as follows:

$$\begin{aligned}&\text{Find vector}\\&\mathrm{X}=(x_1,...,x_D),\quad X\in\Re^D\\&\text{to minimize objective function}\\&f(X)\\&\text{subject to constraint functions}\\&g_j(X)\le 0\qquad j=1,...,m\\&\text{and subject to boundary constraints}\\&x_i^{(L)}\le x_i\le x_i^{(U)}\quad i=1,...,D\end{aligned}\tag{1}$$

Thus, the optimization goal is to find a feasible vector *X*, composed of *D* parameters, to minimize the *objective function f(X).*

B. The Algorithm

As with all evolutionary optimization algorithms, DE operates on a *population*, P_G, of candidate solutions, not just a single solution. These candidate solutions are the *individuals* of the population. In particular, DE maintains a population of constant size that consists of *NP*, real-valued vectors, $X_{i,G}$, where *i* indexes the population and *G* is the *generation* to which the population belongs.

$$P_G=(X_{1,G},...,X_{NP,G})\qquad G=0,...,G_{max}\tag{2}$$

Additionally, each vector contains *D* real parameters:

$$X_{i,G}=(x_{1,i,G},...,x_{D,i,G})\qquad i=1,...,NP,\ G=0,...,G_{\max}\tag{3}$$

For establishing a starting point for optimum seeking, the population must be initialized. Often there is no more knowledge available about the location of a global optimum than the limits of the problem variables. Then a natural way to seed the initial population, $P_{G=0}$, is with random values chosen from within the given boundaries:

$$\begin{aligned}&x_{j,i,0}=rnd_j[0,1]\cdot\left(x_j^{(U)}-x_j^{(L)}\right)+x_j^{(L)}\\&i=1,...,NP,\quad j=1,...,D\end{aligned}\tag{4}$$

where $rnd_j[0,1]$ denotes a uniformly distributed random value within the range: [0.0,1.0] that is chosen anew for each *j*.

DE's self-referential population reproduction scheme is different from other evolutionary algorithms. From the 1st generation forward, vectors in the current population, P_G, are randomly sampled and combined to create candidate vectors for the subsequent generation, P_{G+1}. The population of candidate, or "trial" vectors $P'_{G+1} = U_{i,G+1} = u_{j,i,G+1}$ (where $i = 1,\ldots,NP$, $j = 1,\ldots,D$), is generated as follows:

$$\begin{aligned}&u_{j,i,G+1}=\begin{cases}v_{j,i,G+1} & \text{if } rnd_j[0,1)\le CR\vee j=k\\ x_{j,i,G} & \text{otherwise}\end{cases}\\&\text{where}\\&v_{j,i,G+1}=x_{j,r3,G}+F\cdot\left(x_{j,r1,G}-x_{j,r2,G}\right)\\&i=1,...,NP,\quad j=1,...,D\\&k_i\in\{1,...,D\},\text{ random parameter index}\\&r_1,r_2,r_3\in\{1,...,NP\},\text{ randomly selected,}\\&\text{except}: r_1\ne r_2\ne r_3\ne i\\&CR\in[0,1],\quad F\in(0,1+]\end{aligned}\tag{5}$$

The randomly chosen indexes, r_1, r_2 and r_3 are different from each other and also different from the running index, *i*. New, random, integer values for r_1, r_2 and r_3 are chosen for each value of the index *i*, i.e., for each individual. The index, *k*, refers to a randomly chosen chromosome which is used to ensure that each individual trial vector, $U_{i,G+1}$, differs from its counterpart in the previous generation, $X_{i,G}$, by at least one parameter. A new, random, integer value is assigned to *k* prior to the construction of each trial vector, i.e., for each value of the index *i*.

F and *CR* are DE control parameters. Like *NP*, both values remain constant during the search process. *F* is a real-valued factor in the range (0.0,1.0+] that scales the differential variations, and therefore controls mutation amplifications. Respectively, *CR* is a real-valued crossover factor in range [0.0,1.0] that controls the probability that a trial vector parameter will come from the randomly chosen, mutated vector, $v_{j,i,G+1}$, instead of from the current vector, $x_{j,i,G}$.

Usually, suitable values for *F*, *CR* and *NP* can be found by trial-and-error after a few tests using different values. Practical advice on how to select control parameters *NP*, *F* and *CR* can be found in [9, 12, 13], for example.

DE's replacement scheme also differs from other evolutionary algorithms. The population for the next generation, P_{G+1}, is selected from the current population, P_G, and the child population, according to the following rule:

$$X_{i,G+1}=\begin{cases}U_{i,G+1} & \text{if } f(U_{i,G+1})\le f(X_{i,G})\\ X_{i,G} & \text{otherwise}\end{cases}\tag{6}$$

Thus, each individual of the temporary population is compared with its counterpart in the current population. Assuming that the objective function is to be minimized, the vector with the lower objective function value wins a place in the next generation's population. As a result, all the individuals of the next generation are as good or better than their counterparts in the current generation. The interesting point concerning DE's replacement scheme is that a trial vector is only compared to one individual, not to all the individuals in the current population.

C. Handling Boundary Constraints

In boundary constrained problems, it is essential to ensure that parameter values lie inside their allowed ranges after reproduction. A simple way to guarantee this is to replace parameter values that violate boundary constraints with random values generated within the feasible range:

$$u_{j,i,G+1} = \begin{cases} rnd_j[0,1) \cdot \left(x_j^{(U)} - x_j^{(L)}\right) + x_j^{(L)} \\ \text{if } \; u_{j,i,G+1} < x_j^{(L)} \vee u_{j,i,G+1} > x_j^{(U)} \\ u_{j,i,G+1} \quad \text{otherwise} \end{cases} \tag{7}$$

where

$i = 1,\ldots,NP, \quad j = 1,\ldots,D$

This is the method that was used for this work. However, also some alternative approaches for handling boundary constraints can be found in the literature [7].

It should also be mentioned that DE's generating scheme can extend its search beyond the space defined by initial parameter limits (Eq. 4 and 5) if allowed to do so. This can be a beneficial property for unconstrained problems because optima that are outside the initialized range can often be located.

D. Handling Constraint Functions

In the literature [7], mostly various penalty function methods have been applied with DE for handling constraint functions. In its simplest form, the function value $f'(X)$ to be minimized by DE can be computed by penalizing the objective function value with a weighted sum of constraint violations, for example as follows:

$$f'(X) = f(X) + \sum_{j=1}^{m} w_j \cdot \max\left(0, g_j(X)\right) \tag{8}$$

The penalty function approach effectively converts a constrained problem into an unconstrained one, and then $f'(X)$ is used instead of $f(X)$ as objective function.

Typically a penalty function method will establish one or more additional control parameters, penalty parameters or the weights like the ones in Eq. 8. Appropriate values for the control parameters are expected to be set by the user *a priori*. In practice this stage always requires some additional efforts, and often finding suitable setting may be laborious or even impossible task in practice. Quite often several trial runs of the DE were required for setting the penalty parameters by trial-and-error. The need for developing improved constraint handling techniques is obvious.

III. NEW APPROACH FOR CONSTRAINT HANDLING

The proposed modification for DE's replacement rule is substituting the current rule (Eq. 6) with the following one:

$$X_{i,G+1} = \begin{cases} U_{i,G+1} \quad \text{if} \quad \begin{cases} \forall j \in \{1,\ldots,m\}: g_j(U_{i,G+1}) \le 0 \wedge g_j(X_{i,G}) \le 0 \\ \wedge \\ f(U_{i,G+1}) \le f(X_{i,G}) \end{cases} \\ \qquad\qquad \vee \\ \qquad\qquad \begin{cases} \forall j \in \{1,\ldots,m\}: g_j(U_{i,G+1}) \le 0 \\ \wedge \\ \exists j \in \{1,\ldots,m\}: g_j(X_{i,G}) > 0 \end{cases} \\ \qquad\qquad \vee \\ \qquad\qquad \begin{cases} \exists j \in \{1,\ldots,m\}: g_j(U_{i,G+1}) > 0 \\ \wedge \\ \forall j \in \{1,\ldots,m\}: g'_j(U_{i,G+1}) \le g'_j(X_{i,G}) \end{cases} \\ X_{i,G} \quad \text{otherwise} \end{cases} \tag{9}$$

where

$g'_j(X_{i,G}) = \max\left(g_j(X_{i,G}), 0\right)$

$g'_j(U_{i,G+1}) = \max\left(g_j(U_{i,G+1}), 0\right)$

Thus, when compared with the corresponding member, $X_{i,G}$, of the current population, the trial vector, $U_{i,G+1}$, will be selected if:

- it satisfies all constraints and provides a lower or equal objective function value, while both the compared solutions are feasible, or
- it is feasible while $X_{i,G}$ is infeasible, or
- it is infeasible, but provides a lower or equal constraint violation for all constraint functions.

Note that in case of an infeasible solution, the replacement rule does not compare the objective function values. Thus no selective pressure exists towards the search space regions providing low objective values, but infeasible solutions. Instead, a selective pressure towards regions, where constraint violation decreases, will be present in general. Because of that, also for finding the first feasible solution an effective selection pressure will be applied. This result in a fast convergence to the feasible regions of the search space, as demonstrated in the following Section.

Note also, that DE's replacement operation does not essentially require numerical objective function values for performing the selection, but only the result of the comparison; Which one of the compared solutions is better? Here it is considered that:

- If both the compared solutions are feasible, the one with lower objective function value is better.
- Feasible solution is better than infeasible.
- If both the compared solutions are infeasible, the situation is less obvious. Then the candidate vector can be considered less infeasible, and thus better than

the current vector, if it does not violate any of the constraints more than the current vector and if it violate less at least one of the constraints.

The last mentioned concept for comparing two infeasible solutions is analogous with, and derived from, the concepts of Pareto-optimality. The selection is based on Pareto-dominance in the effective constraint function space, $g'(X) = \max(0 , g(X))$. Note, that the Pareto-optimal front in the effective constraint function space is a single point ($g'(X) = 0$), if a feasible solution for the problem exists.

In addition, the candidate vectors considered equally good as the compared current population member are allowed to enter into the population in order to avoid stagnation [9], to flat regions of the objective function surface.

The proposed approach does not introduce any extra search parameters to be set by the user – there are no constraint handling related, or any other search parameters in Eq.9.

An important special case should also be noted here. In case of an unconstrained problem ($m = 0$), Eq. 9 effectively reduces into the DE's original replacement rule (Eq. 6). Thus the new replacement rule does not change the DE from this point of view, except extending its capabilities for handling constraints.

IV. NUMERICAL EXPERIMENTS

The proposed method was tested with a set of ten benchmark problems, often referred as *Michalewicz's problem set* [6, 11] , despite the problems have been originally taken elsewhere [2, 3, 4]. The properties of test problems are summarized briefly in Table I. The details for all ten test problems can be found from [6, 11] and via Internet: `http://www.lut.fi/~jlampine/testset.pdf` .

For each test problem, total 1000 independent runs of the algorithm were performed in order to assess the feasibility, performance and robustness of the method. For each problem, the worst, the average and the best objective function values out of 1000 independent runs were recorded. Also the number of objective function evaluations for obtaining the first feasible solution and the processor time used for computation were recorded.

The search parameters of DE were set to $F = 0.9$ and $CR = 0.9$ for all problems. Only the population size and the number of generations were coarsely varied to provide some problem specific adaptation. For each problem, the population size N_P was selected among the multiples of 5 and the number of generations was selected from the multiples of 500. These selections were based on the author's experience and the recommendations given in [9]. Any further settings, like penalty parameters, weights for each constraint function or extra search parameters, were not required. No particular attempts were performed in order to optimize the search parameter settings to suit for each test problem. The applied DE's control parameter settings for each problem are summarized in Table II.

In order to avoid converting effectively the problems including equality constraints (problems 8, 9 and 10) into a multiobjective problem, each equality constraint were substituted with two inequalities, allowing a small tolerance for equality constraint violation, e.g. $g(X) = 0$ was substituted with $-0.001 \le g(X) \le 0.001$.

The proposed method was implemented with Compaq Visual Fortran 6.1 environment, and a PC with Pentium III processor at 500MHz was applied for all computations.

TABLE I
SUMMARY OF THE TEST PROBLEMS

Problem	Number of Variables	Objective Function	Number of Constraint Functions [1]			Problem no. in [6]
			Total	Linear	Nonlinear	
1	13	nonlinear	9	9	–	G1
2	7	nonlinear	4	–	4	G9
3	10	nonlinear	8	3	5	G7
4	8	linear	6	3	3	G10
5	2	nonlinear	2	–	2	G8
6	2	nonlinear	2	–	2	G6
7	5	nonlinear	6	–	6	G4
8	4	nonlinear	5 (8)	2 (2)	3 (6)	G5
9	2	nonlinear	1 (2)	–	1 (2)	G11
10	50	nonlinear	50 (100)	–	50 (100)	G3

[1] Each equality constraint function was substituted with two inequality constraints requiring $-0.001 \ge g(X) \ge 0.001$. Value in brackets refers to the number of constraints after this conversion. Other constraints were required to be satisfied strictly.

TABLE II
CONTROL PARAMETER SETTINGS

Problem	Mutation parameter, F	Crossover parameter, C_r	Population size, N_p	Number of generations	Number of function evaluations
1	$F = 0.9$ for all problems	$C_r = 0.9$ for all problems	20	4000	80000
2			25	1500	37500
3			35	5000	175000
4			30	9000	270000
5			20	500	10000
6			15	1000	15000
7			15	2000	30000
8			120	100000	12000000
9			30	1000	30000
10			40	200000	8000000

V. RESULTS

In Table III the required number of function evaluations for finding the first feasible solution were reported and compared with the results obtained by applying pure random search. As expected, the results indicate that the proposed replacement method introduces effectively a selective pressure towards the feasible region of the search space while no feasible solutions have been found yet. The proposed method does not need a feasible initial solution as a starting point – and it appears to be capable of searching efficiently for feasible solutions itself.

Note from the Table III, that the inverse $1/FE_{rs}$ of the average number of function evaluations for finding a feasible solution by random search estimates the size of the feasible region, F, in relation to the entire search space, S. In cases for the applied test problems, the feasible set represents from almost 0% to 52.0% of the search space. The most difficult

problems in the test problem set, problems 1, 3, 4, 6, 8 and 10, are heavily constrained problems having all ratio *F*/*S* less than 10^{-4}. For the most difficult problems, 8 and 10, the random search did not found any feasible solution within reasonable CPU-time (more than a week of CPU-time used).

The results in Table III suggest, that the proposed method is much more efficient in searching the first feasible solution than the random search. For the tested problems DE found the first feasible solution 3% – 99% faster than the random search. The difference was the highest with the most difficult test cases. For the problems 1, 3, 4, 6, 8 and 10, the proposed method found the first feasible solution 95% – 99% faster than the random search. This is rather expected result, since in case that almost the whole search space is feasible, the difference cannot actually be higher because virtually any method is likely to find a feasible solution by its first few trials. However, in cases of the more constrained problems, searching a feasible solution randomly will soon turn out hopelessly inefficient, like in cases for problems 8 and 10.

The justification for initial comparisons with the random search is that the random search is often applied for searching a feasible starting point for the methods requiring it. As well, it was relevant to find out, whether the proposed method is, in fact, any more efficient in finding the first feasible solution than the random search. Furthermore, the results of the random search make it possible to estimate the relative size of the feasible region, ratio *F*/*S* , for each test problem.

Table IV summarizes the experimental results for optimizing the test problems with the proposed method and compares the results with the best results reported in [5, 6, 10]. For each test problem, only the best of the compared methods are included in Table IV.

TABLE III
NUMBER OF FUNCTION EVALUATIONS FOR FINDING THE FIRST FEASIBLE SOLUTION.

Problem	Differential Evolution [1]			Random Search [1]	Ratio	Ratio
	Worst FE_w	Average FE_{avg}	Best FE_b	Average FE_{rs}	F/S	FE_{avg}/FE_{rs}
1	6409	3049	265	382946	2.61×10^{-6}	0.8%
2	464	103	1	194	5.12×10^{-3}	53%
3	23576	9809	69	881751	1.13×10^{-6}	1.1%
4	20589	8307	382	166479	6.01×10^{-6}	5.0%
5	375	84	1	112	8.93×10^{-3}	75%
6	645	350	12	15076	66.3×10^{-6}	2.3%
7	9	1.88	1	1.93	0.52	97%
8	11490917	2680623	1076210	160×10^{-9}	– [2]	0%
9	2672	525	1	983	1.02×10^{-3}	53%
10	228917	121561	77994	17.8×10^{-9}	– [2]	0%

[1] Results are based on 1000 independent runs with both methods.
[2] No feasible solution found by the random search despite the reported high number of function evaluations (requiring more than a week of CPU time).

TABLE IV
SUMMARY OF THE EXPERIMENTAL RESULTS AND COMPARISONS

Problem	Solution by	Number of Function Evaluations	Processor Time	Optimal or Best Known Solution f(X*)	Found Solution			Error in Worst Case $f_w(X)-f(X^*)$	Average Error $f_a(X)-f(X^*)$
					Worst $f_w(X)$	Average $f_a(X)$	Best $f_b(X)$		
1	DE [1]	80000	0.57 sec	-15.000 [3]	-15.000	-15.000	-15.000	0.000	0.000
	[10] [2]	n/a	n/a		n/c	-15.000	n/c	n/c	0.000
2	DE [1]	37500	0.17 sec	680.630 [4]	680.630	680.630	680.630	0.000	0.000
	[10] [2]	n/a	n/a		n/c	680.718	n/c	n/c	0.088
3	DE [1]	175000	1.08 sec	24.306 [4]	24.307	24.306	24.306	0.001	0.000
	[6] [2]	n/a	n/a		n/c	24.826	n/c	n/c	0.520
4	DE [1]	270000	2.84 sec	7049.248 [5]	7049.248	7049.248	7049.248	0.000	0.000
	[6] [2]	n/a	n/a		n/c	7498.6	n/c	n/c	449.4
5	DE [1]	10000	0.04 sec	0.095825	0.095825	0.095825	0.095825	0.000	0.000
	[6] [2]	n/a	n/a		n/c	0.095825	n/c	n/c	0.000
6	DE [1]	15000	0.05 sec	-6961.814 [3]	-6961.814	-6961.814	-6961.814	0.000	0.000
	[6] [2]	n/a	n/a		n/c	-6948.1	n/c	n/c	13.7
7	DE [1]	30000	0.22 sec	-31025.6 [5]	-31025.6	-31025.6	-31025.6	0.000	0.000
	[6] [2]	n/a	n/a		n/c	-30655.3	n/c	n/c	370.3
8	DE [1] [6]	12000000	68.5 sec	5126.4981	5126.484	5126.484	5126.484	0.014	-0.014
	[5] [2] [6]	n/a	n/a		n/c	5207.9604	n/c	n/c	81.46
9	DE [1] [6]	30000	0.07 sec	0.75000455	0.74900	0.74900	0.74900	0.001	-0.001
	[6] [2] [6]	n/a	n/a		n/c	0.75	n/c	n/c	0.000
10	DE [1] [6]	8000000	195 sec	-1.0000	-1.0252	-1.0252	-1.0252	0.0252	-0.0252
	[6] [2] [6]	n/a	n/a		n/c	-0.9989	n/c	n/c	0.0011

[1] Results are based on 1000 independent runs with DE.
[2] Results are based on only 10 independent runs of the compared algorithms.
[3] Reported by Floudas & Pardalos [2].
[4] Reported by Hock & Schittkowski [4].
[5] Found by DE in this investigation.
[6] Small equality constraint violation was allowed while converting them into two inequalities: $-0.001 \le g(X) \le 0.001$.
n/a – not available.
n/c – not comparable (different number of test runs performed).

0-7803-7282-4/02/$10.00 ©2002 IEEE

For all test problems, the proposed method found the best known solution, except for the problems 4 and 7. For these problems even better solutions were found, than the best known solution reported (so far) in the literature. Note that in case of the proposed method, 1000 independent optimization runs were performed for each test problem. In all cases the reported optimum solution were found.

As reported in Table IV, the CPU time for solving each test problems varied from a few seconds to a few minutes using a PC with Pentium III-500MHz processor. The computation times for the compared results were not reported in [5, 6, 10]. However, CPU times ranging from 1 hour to 8 hours for solving the same problems with Genetic Algorithm and Sun Sparc/20 processor are reported in [1]. This information enables some rough and preliminary comparisons, since the performances of the compared processors are within the same order of magnitude.

Another important advantage of the proposed approach is, that evaluation of a trial solution require evaluation of all functions involved only in case that both compared solutions, $X_{i,G}$ and the current population member $U_{i,G+1}$, are both feasible (see Eq. 9). For example, in case for an infeasible trial solution, the objective function value is not needed for performing the selection. Respectively, in case of comparing two infeasible solutions, the current population member, $X_{i,G}$, can be selected immediately after finding that the trial, $U_{i,G+1}$, violates any of the constraints more than $X_{i,G}$. Evaluation of the remaining constraint functions is not necessary. Taking advantage of this possibility, we performed an experiment in case of the test problem 4, in which this modification resulted in 87% CPU time reduction (from 2.84 sec to 0.17 sec) in comparison with the case for evaluating all functions involved for every trial solution.

VI. CONCLUSIONS

In this article an extension of the DE algorithm for handling multiple constraint functions was proposed and demonstrated with a set of ten well-known test problems. In comparison with the original DE algorithm, only the replacement operation was modified by applying a new replacement criterion for handling the constraint functions.

With all test cases studied here, the algorithm demonstrated effectiveness, efficiency and robustness. All the studied problems were solved using only 0.04–195 seconds processor time of a PC. All performed 1000 independent tests for each test case returned practically identical results without failures.

For the experiments, the search parameters of the DE were selected rather roughly. No particular attempts were performed towards optimizing the problem specific adaptation – only the population size and the number of generations was varied from a problem to another. Still the results in all cases were constantly equal or better than the compared results.

The proposed approach is straightforward to use from the user's point of view, allowing solving multi-constrained problems by DE virtually as easily as unconstrained ones. Providing a feasible a starting point or tuning extra control parameter settings are not required. These are considered as the most important contributions of this article because of their practical relevance for simplifying many real-life problem-solving situations by providing a convenient way to handle multiple constraint functions.

Another major advantage is the possibility for considerable CPU timesaving due to fact, that evaluating a trial solution does not always require evaluating the objective function and all constraint functions involved.

Despite of the encouraging results with the current test problems, due to limited size of the test problem set, only drawing preliminary conclusions is justified concerning the effectiveness, efficiency and robustness of the proposed method. Further research in future will be essential. However, the current results certainly justify and motivate the further work with the proposed method.

REFERENCES

[1] Camponogara, E. and Talukdar, S.N. (1997). *A Genetic Algorithm for Constrained and Multiobjective Optimization*. In: Alander, J.T. (ed.). Proc. of the 3nd Nordic Workshop on Genetic Algorithms and their Applications, Helsinki, Finland, August 1997, pp. 49–61. Finnish Artificial Intelligence Society.

[2] Floudas, C.A. and Pardalos, P.M. (1987). *A Collection of Test Problems for Constrained Global Optimization Algorithms.* Lecture Notes in Computer Science, vol. 455. Springer-Verlag.

[3] Himmelblau, D. (1992). *Applied Nonlinear Programming.* McGraw-Hill.

[4] Hock, W. and Schittkowski, K. (1981). *Test Examples for Nonlinear Programming Codes*. Lecture Notes in Economics and Mathematical Systems, vol. 187. Springer-Verlag. ISBN 3-540-10561-1 and ISBN 0-387-10561-1.

[5] Joines, J.A. and Houck, C.R. (1994). *On the Use of Non-Stationary Penalty Functions to Solve Nonlinear Constrained Optimization Problems.* In: Proceedings of the First IEEE Conference on Evolutionary Computation, 27.-29. June 1994, vol. 2, pp. 579-584. ISBN 0-7803-1899-4.

[6] Koziel, S. and Michalewicz, Z. (1999). *Evolutionary Algorithms, Homomorphous Mappings, and Constrained Parameter Optimization*. Evolutionary Computation 7(1):19-44, 1999.

[7] Lampinen, J. (2001). *A Bibliography of Differential Evolution Algorithm*. Technical Report. Lappeenranta University of Technology, Lab. of Information Processing. Available via Internet: `http://www.lut.fi/~jlampine/debiblio.htm` . Cited 30th November 2001.

[8] Lampinen, J. and Zelinka, I. (1999). *Mechanical Engineering Design Optimization by Differential Evolution*. In: Corne, D., Dorigo, M. and Glover, F. (eds.). New Ideas in Optimization, pp. 127–146. McGraw-Hill, London. ISBN 007-709506-5.

[9] Lampinen, J. and Zelinka, I. (2000). *On Stagnation of the Differential Evolution Algorithm*. In: Ošmera, P. (ed.). Proc. of MENDEL 2000, 6th Int. Conf. on Soft Computing, June 7.–9. 2000, Brno University of Technology, Brno, Czech Republic, pp. 76–83. ISBN 80-214-1609-2.

[10] Michalewicz, Z. (1995). *Genetic Algorithms, Numerical Optimization and Constraints*. In: Proceedings of the 6th International Conference on Genetic Algorithms, Pittsburgh, July 15.–19. 1995, pp. 151-158.

[11] Michalewicz, Z. and Shoenauer, M. (1996). *Evolutionary Algorithms for Constrained Parameter Optimization Problems*. Evolutionary Computation 4(1):1-32, 1996.

[12] Price, K.V. (1999). *An Introduction to Differential Evolution*. In: Corne, D., Dorigo, M. and Glover, F. (eds.). New Ideas in Optimization, pp. 79–108. McGraw-Hill, London. ISBN 007-709506-5.

[13] Storn, R. and Price, K.V. (1997). *Differential Evolution – a Simple and Efficient Heuristic for Global Optimization over Continuous Spaces*. Journal of Global Optimization, 11(4):341–359, December 1997. Kluwer Academic Publishers.