Cutting Box Strategy: an algorithmic framework for improving metaheuristics for continuous global optimization

Wendel Melo[1,2]*, ***Marcia Fampa***[1,2], ***Fernanda Raupp***[3]

1 - Institute of Mathematics, Federal University of Rio de Janeiro, Brazil.
2 - Alberto Luiz Coimbra Institute - Graduate school and Research in Engineering (COPPE), Federal University of Rio de Janeiro, Brazil.
3 - National Laboratory for Scientific Computing (LNCC), Brazil.

Abstract

In this work, we present a new framework to increase effectiveness of metaheuristics in seeking good solutions for the general nonlinear optimization problem, called *Cutting Box Strategy* (CBS). CBS is based on progressive reduction of the search space through the use of intelligent multi-starts, where solutions already obtained cannot be revisited by the adopted metaheuristic. Computational experiments with the CBS strategy are conducted with a variant of the population-based metaheuristic Differential Evolution to solve 36 test instances. The numerical results show that CBS can substantially increase the quality of the results of a metaheuristic applied for a nonlinear optimization problems.

Keywords: Metaheuristic, Constrained Optimization, Continuous Global Optimization, Differential Evolution.

1 Introduction

Continuous global optimization problems arise in several practical applications related to natural, exact and economic sciences. In a general form, the

*E-mail address: wendelmelo@cos.ufrj.br

problem can be express as finding a $x^* \in \mathbb{R}^n$ that solves:

$$\begin{aligned} \text{minimize}_x \quad & f(x) & & (1)\\ \text{subject to} \quad & g_i(x) \le 0, \quad i = 1, \ldots, p, & & (2)\\ & h_j(x) = 0, \quad j = 1, \ldots, q, & & (3)\\ & l_k \le x_k \le u_k, \quad k = 1, \ldots, n, & & (4) \end{aligned}$$

where f, g_i, $i = 1, \ldots, p$, and h_j, $j = 1, \ldots, q$ are real-valued functions, not necessarily convex neither continuous nor differentiable, and l and u are n-dimensional vectors denoting lower and upper bounds to variables, respectively, used to define the *box constraints* (4). Denoting the feasible region of problem (1) by $\mathcal{F}$, we say that $\bar{x} \in \mathcal{F}$ is a local optimal solution if $f(\bar{x}) \le f(x)$ for all $x \in \mathcal{F}$ in a neighborhood of $\bar{x}$. A solution $x^* \in \mathcal{F}$ such that $f(x^*) \le f(x)$ for all $x \in \mathcal{F}$ is denoted global optimal solution, which is the solution of interest.

Classical continuous methods that address problem (1) may have trouble in finding some of their global optimal solutions, converging to a local optimal solution, or even to a solution that is neither a local nor a global optimal. In some cases, these methods cannot achieve a feasible solution, even though the feasible region is non-empty. This difficulty in finding a global optimal solution to the addressed problem can become critical if the problem (1) is not convex, or any of its objective or constraint functions is not continuous or not differentiable at some point in $\{x^* \in \mathbb{R}^n \mid l \le x \le u\}$.

To circumvent these difficulties, many researchers have developed studies on the application of stochastic local search procedures or metaheuristics that solve problem (1) (see, for example, [1, 2, 4, 10–13, 15, 17–19, 21–23, 25, 27–29, 32]). Metaheuristics are nondeterministic procedures that make use of randomness in a variety of strategies in order to avoid premature convergence to local optimal solutions. Thereby, it is expected that when applying a metaheuristic to solve problem (1) we would obtain a global optimal solution, or at least a solution close to it. The quality of the solution is, in many cases, closely related to the efficacy of the algorithm in exploring the search region in a diverse way and escaping from any local optima possibly found.

Differential Evolution (DE) is one of the most successful metaheuristic approaches for continuous optimization problems [5, 6, 24, 26, 31]. Its general idea is to use a population of candidate solutions that evolve over an iterative process searching for good solutions with the application of crossover and mutation operations. Proposed by Storn and Price in [30], the original algorithm was developed for box constrained problems (i.e., optimization problems with box constraints only). Then, many researches extended its application to solve problem (1) using a variety of strategies [2, 11, 13, 15, 17, 21, 23, 27, 32, 33]. For example, in [17], Melo et al. propose the Differential Evolution Variant (DEV), that adopts advanced techniques to solve problem (1) more effectively, such as adaptive parameters, dynamic update of the population, multiple offspring and a stopping criterion that detects the convergence of the population to a feasible solution of the problem, which is an alternative to the maximum number of iterations, originally proposed.

In this work, we propose a new mechanism, named *Cutting Box Strategy*, to increase the efficiency of local search procedures and metaheuristics on the solution of problem (1), especially when there are multiple local optima. We incorporate this strategy to the already proposed algorithm DEV and evaluate the results for a set of benchmark instances. Although in this work we only evaluate the use of the Cutting Box Strategy for DEV, this strategy is independent from the DE algorithm and can be combined with any other population-based metaheuristic or stochastic local search algorithm that has some mechanism for detecting convergence.

This chapter is organized as follows: Section 2 describes the DEV algorithm, whereas Section 3 presents the Cutting Box Strategy. Computational results are shown in Section 4. Finally, Section 5 presents the conclusions of this work.

2 Differential Evolution Variant

2.1 Differential Evolution

In [30], Storn and Price proposed the Differential Evolution (DE) method, a population-based metaheuristic for continuous optimization problems considering only box constraints. The main idea of this approach is to use a population of candidate solutions that evolve over an iterative process to find the best possible value for the objective function. Given the current population at iteration g, a new descendant solution is generated for each candidate solution of this population. If the descendant solution proves to be better than its respective ascendent solution (target solution), then it will take the place of the former. Otherwise, the descendant solution is discarded and the target solution remains in the population.

In the original DE algorithm, only a single descendant solution is generated for each candidate solution of the current population at each iteration (or generation). However, some DE competitive approaches (e.g., [23, 32]) consider multiple descendant solutions, which consists in generating M descendant solutions for each candidate solution of the population at each generation. If the best solution among the M descendant solutions is better than its respective target solution, then it will take the place of the former generated solution in the population (in this situation, the other $M-1$ descendant solutions are discarded along with the target solution). Otherwise, the target solution remains in the population and the M descendant solutions are discarded. Algorithm 1 shows DE with multiple offspring based on the variation of DE known as DE rand/1/bin [30]. Note that all solutions of the population are used, each in turn, as a target solution. For generating new candidate solutions this approach employs crossover and mutation operators, lines 9-12, where three other random solutions besides the target solution are used. Although the offspring are generated from four solutions, they compete for entry into the population only with the target solution. In general, when a coordinate of a generated descendant solution does not satisfy the box constraint (4), a new random value is generated between its limits. As shown in Algorithm 1, only the maximum number of iterations (*MAXGEN*) is used as a stopping criterion, likewise the original DE approach.

2.2 Differential Evolution Variant aspects

Since the original DE algorithm was developed to minimize functions subject to box constraints, the criterion presented to select the best solution among the target solution and its descendants in line 13 of Algorithm 1, is to choose the solution with the lowest value for the objective function. The DE approaches that address problem (1), however, must adopt more sophisticated selection criteria to deal with the feasibility of the solutions (the interested reader can find a good discussion in [20]). The Differential Evolution Variant (DEV) algorithm, which is used as the basic approach in this work, has as part of its selection criteria all three Deb's comparison criteria [7], listed as follows:
(1) any feasible solution is better than any infeasible one;
(2) between two feasible solutions, the solution with lower objective function value is preferred;
(3) between two infeasible solutions, the solution with lower penalty term is better.
The Deb criteria were also used in many other works, such as, for example, [17–19, 23, 29, 32].

Input: *MAXGEN*: number of generations, P: size of population, M: number of descendants, *CR*: crossover parameter.

Generate and evaluate an initial population $x^{i,0}, i = 1, \ldots, P$;
Set F as a real value between 0 and 1 ;
for $g = 1, \ldots, MAXGEN$ **do**
  for $k = 1, \ldots, P$ **do**
    for $i = 1, \ldots, M$ **do**
      Randomly select $r_1 \neq r_2 \neq r_3 \neq k \in \{1 \ldots P\}$;
      $r_{nbr} \leftarrow randint(1, n)$;
      for $j = 1, \ldots n$ **do**
        if $rand(0, 1) \leq CR$ **or** $j = r_{nbr}$ **then**
          $d_j^{k,i} \leftarrow x_j^{r_3,g} + F(x_j^{r_1,g} - x_j^{r_2,g})$;
        else
          $d_j^{k,i} \leftarrow x_j^{k,g}$;
    Let u^k be the best solution (according to a predefined criterion) among $x^{k,g}$ and $d^{k,i}, i = 1, \ldots, M$;
    $x^{k,g+1} \leftarrow u^k$;

Algorithm 1: DE standard algorithm with multiple descendants.

Algorithm 2 presents the DEV algorithm. We observe that this algorithm adopts multiple offspring and therefore uses a criterion to select one solution among each target solution and its M descendants in every generation (line 15). Based on the method proposed in [19], the selection procedure is presented in Algorithm 3. In this procedure, the parameter S_r defines a probability for the best descendant solution (according to Deb's comparison criteria) to be compared to the target solution only with respect to the objective function, as in the original DE algorithm. Thereby, with probability $1-S_r$ these two solutions will also be compared according to the Deb's criteria. The algorithm can then incorporate "good infeasible solutions" that lead to promising regions within the feasible region. Values near 1.0 for this parameter can provide diversity in the exploration process, but may hinder the location of feasible solutions and the convergence of the algorithm. Therefore, unlike what is proposed in [19], DEV adopts dynamic updating for this parameter in line 14 of Algorithm 2. Notice that in early iterations, the value of S_r is close to S_r^0, and as the algorithm evolves the value assigned to S_r approaches 0.0. In this way, it is possible to properly include diversity at the beginning of the exploration process, and thereafter favor the convergence of the algorithm to a good feasible solution.

In DEV algorithm, parameter CR plays a role similar to S_r. This parameter defines the probability for each coordinate of the descendant solutions to be either equal to their corresponding mutant solution coordinate (line 11) or target solution coordinate (line 13). Intuitively, we could expect that values close to 0.5 would favor diversity in the exploration of the search space. However, empirical studies, e.g. [9, 32], show that values close to 0.9 lead to better performance of the DE algorithm. For this reason, at each iteration DEV updates CR according to the expression in line 3 of Algorithm 2, so that values close to CR^0 are considered at early iterations, and from iteration $\frac{MAXGEN}{2}$, CR is equal to 1.0.

Input: $MAXGEN$: number of generations, P: size of population, M: number of descendants, CR^0: initial crossover parameter, S_r^0: initial selection parameter, ϵ: convergence tolerance.

1 Generate and evaluate a initial population $x^i, i = 1, \ldots, P$;
2 **for** $g = 1, \ldots, MAXGEN$ **do**
3     $CR \leftarrow \min\{(1 - CR^0)\frac{2g}{MAXGEN} + CR^0\ ,\ 1\}$;
4   **for** $k = 1, \ldots, P$ **do**
5   **for** $i = 1, \ldots, M$ **do**
6   Randomly select $r_1 \neq r_2 \neq r_3 \neq k \in \{1 \ldots P\}$;
7             $r_{nbr} \leftarrow randint(1, n)$;
8             $F \leftarrow rand(0.3, 0.9)$;
9   **for** $j = 1, \ldots, D$ **do**
10   **if** $rand(0,1) < CR$ **or** $j = r_{nbr}$ **then**
11                     $d_j^{k,i} \leftarrow x_j^{r_3} + F(x_j^{r_1} - x_j^{r_2})$;
12   **else**
13                     $d_j^{k,i} \leftarrow x_j^k$;
14         $S_r \leftarrow S_r^0 * (1 - \frac{g}{MAXGEN})$;
15         $u^k \leftarrow SelectionOp(S_r, x^k, d^k)$;
16         $x^k \leftarrow u^k$;
    `// Alternative stopping rule`
17     $\overline{\sigma}_i \leftarrow \max(x_i^1, x_i^2, \ldots, x_i^P),\ i = 1, \ldots, n$;
18     $\underline{\sigma}_i \leftarrow \min(x_i^1, x_i^2, \ldots, x_i^P),\ i = 1, \ldots, n$;
19   **if** $\|\overline{\sigma} - \underline{\sigma}\|_\infty < \epsilon$ **and** *all current population solutions are feasible* **then**
20   stop the evolutionary process ;

Algorithm 2: DEV Algorithm.

Also note that at line 16 of Algorithm 2, the solution x^k of the population is updated and becomes immediately available for the crossover and mutation operators, still in the current iteration, unlike the original DE algorithm, where the updated solution only becomes available in the next iteration. This strategy had been used before in [3], and according to the authors, it accelerates the convergence of the DE algorithm, besides saving memory in the computational implementation.

Another important feature of the DEV algorithm is the incorporation of a stopping criterion alternative to the maximum number of iterations (rows 17-20). The test stops the evolutionary process if all the solutions in the population are feasible and belong to a hypercube with edge length $\epsilon > 0$. Stopping the algorithm by the detection of convergence, this criterion can save considerable computational effort. We emphasize that, when the population converges, it is not effective to continue the evolutionary process, since each coordinate of new generated solutions depends exclusively on the respective coordinate of the solutions already in the population, regardless of their quality (regarding the objective function value). This ability to detect the convergence of the population was the main motivation for developing the Cutting Box Strategy, which is detailed in the next section.

Input: S_r: selection parameter, x^k: target solution, d^k: descendant solutions of target solution.

Choose u as the best solution among the M descendant generated solutions (d^k) (according to the Deb's comparison criteria) ;
if $rand(0,1) \leq S_r$ **then**
    if $f(u) \leq f(x^k)$ **then**
        $x^s \leftarrow u$;
    else
        $x^s \leftarrow x^k$;
else
    if *u is better than* x^k *(according to the Deb's comparison criteria)* **then**
        $x^s \leftarrow u$;
    else
        $x^s \leftarrow x^k$;
return x^s ;

Algorithm 3: Procedure *SelectionOp*.

3 Cutting Box Strategy

Although good metaheuristics make use of mechanisms to favor the exploration of the search space with diversity, they may fail to find global solutions especially if the addressed problem presents many local optimal points. To increase the effectiveness of metaheuristics in seeking good solutions, we present in this work a new algorithmic framework that aims to prevent premature convergence of the main procedure adopted. This algorithm, called *Cutting Box Strategy* (CBS), can be used with any metaheuristic that has some mechanism to detect convergence, as for example, DEV (lines 17-20 of the Algorithm 2), even if this convergence is neither to a global nor to a local optimal solution.

The main idea of the CBS strategy is to apply a metaheuristic to solve problem 1 several times, in a scheme that can be represented by a tree, the CBS tree (Figure 1). The metaheuristic adopted in the solution of 1 will be referred in the remaining of this paper as the *basic procedure*. At each level w of the tree, CBS adresses a subproblem obtained from (1) by replacing the box constraint (4):

$$l \leq x \leq u,$$

by:

$$l^w \leq x \leq u^w, \tag{5}$$

where l^w and u^w are such that the constraint (5) defines a subset of the feasilble set defined by (4), i.e., any solution $\bar{x}$ that satisfies (5) shall also satisfy (4). We, thereby, define the box $B^w = \{x \in \mathbb{R}^n \mid l^w \leq x \leq u^w\}$.

Initially, at the root node of the CBS tree, $l^0 = l$ and $u^0 = u$ are set. As the algorithm CBS descends in the tree, the box that represents the search space is reduced by a factor denoted by λ and constructed around the best solution obtained on the current level. To avoid bottlenecks around wrong solutions a number of $maxSubBoxes$ calls to the basic procedure are performed on each level w. Each of these calls to the basic procedure with input B^w will result in a different solution to the addressed problem, which is accomplished by the use of the concept of *cutting box*. Let $\bar{x}^{w,z}$ be the solution obtained on the z-th call to the basic procedure on the level w. Considering the solution $\bar{x}^{w,z}$ and the reduction factor λ, define the box:

$$\bar{B}^{w,z} = \{x \in \mathbb{R}^n | l^{w,z} \leq x \leq u^{w,z}\}$$

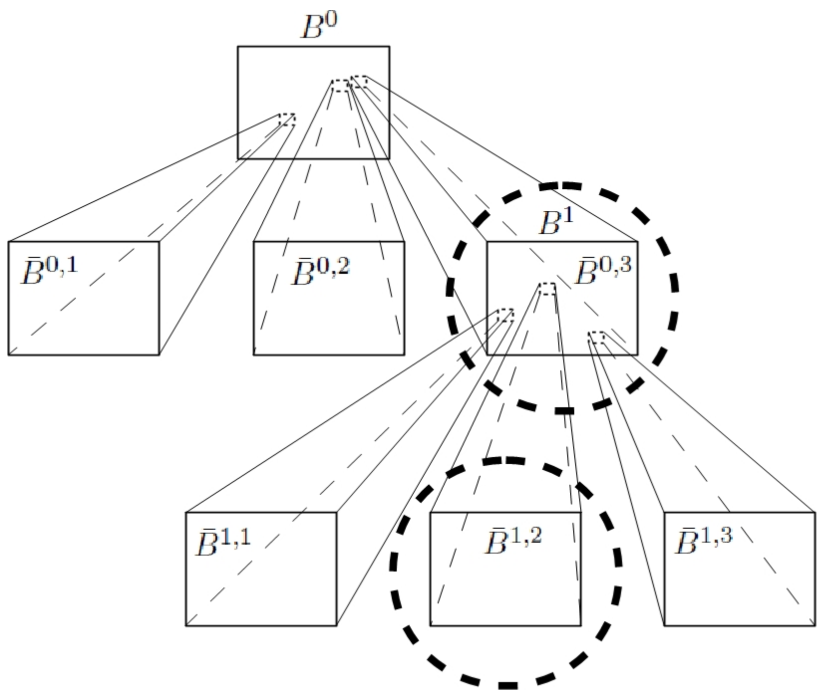

Figure 1: Example of the CBS tree with $maxSubBoxes = 3$ and $maxLevels = 2$. On level zero, box 3 ($\bar{B}^{0,3}$) presents the best solution. On level 1, box 2 ($\bar{B}^{1,2}$) presents the best solution, which is returned by the algorithm (see Algorithm 4).

where

$$l_i^{w,z} = \max\{\bar{x}_i^{w,z} - (u_i^w - l_i^w)\frac{\lambda}{2},\ l_i^w\},\ i = 1, \dots, n \tag{6}$$

$$u_i^{w,z} = \min\{\bar{x}_i^{w,z} + (u_i^w - l_i^w)\frac{\lambda}{2},\ u_i^w\},\ i = 1, \dots, n \tag{7}$$

Input: *BASICPROC*: basic procedure, $maxLevels$: maximum number of levels in a tree, $maxSubBoxes$: maximum number of boxes on each level, α: convergence factor for *BASICPROC*, λ: box reduction factor.

1 $l^0 \leftarrow l,\ u^0 \leftarrow u$;
2 let $B^w = \{x \in \mathbb{R}^n \mid l^w \leq x \leq u^w\}$;
3 $fbest \leftarrow \infty$;
4 **for** $w = 0, \dots, maxLevels - 1$ **do**
5 $numSubBoxes \leftarrow 0$;
6 **for** $z = 1, \dots, maxSubBoxes$ **do**
7   Apply *BASICPROC* to box B^w preventing solutions in boxes $\bar{B}^{w,j}$, $j = 1, \dots, z-1$;
8   **if** *BASICPROC converges* **then**
9    $numSubBoxes \leftarrow numSubBoxes + 1$;
10   let $\bar{x}^{w,z}$ the best solution obtained in line 7 ;
11    $l_i^{w,z} \leftarrow \max\{\bar{x}_i^{w,z} - (u_i^w - l_i^w)\frac{\lambda}{2},\ l_i^w\},\ i = 1, \dots, n$;
12    $u_i^{w,z} \leftarrow \min\{\bar{x}_i^{w,z} + (u_i^w - l_i^w)\frac{\lambda}{2},\ u_i^w\},\ i = 1, \dots, n$;
13    $\bar{B}^{w,z} \leftarrow \{x \in \mathbb{R}^n \mid l^{w,z} \leq x \leq u^{w,z}\}$;
14   **else**
15   go to line 16 (*break the loop*) ;
16 **if** $numSubBoxes > 0$ **then**
17   go to line 23 (*break the loop*) ;
18 $b \leftarrow \text{argmin}_{j=1,\dots,numSubBoxes}\{f(\bar{x}^{w,j})\}$;
19 $l^{w+1} \leftarrow l^{w,b},\ u^{w+1} \leftarrow u^{w,b}$;
20 **if** $f(\bar{x}^{w,b}) < fbest$ **then**
21   $xbest \leftarrow \bar{x}^{w,b}$;
22   $fbest \leftarrow f(\bar{x}^{w,b})$;
23 **return** $xbest$;

Algorithm 4: Cutting Box Strategy Algorithm.

In the $(z+1)$-th call to the basic procedure on level w, this procedure adresses the problem defined over box B^w, preventing the obtained solutions to belong to any of the boxes $\bar{B}^{w,j},\ j = 1, \dots, z$. Regarding the DEV algorithm, this may be accomplished as follows: whenever the algorithm generates a solution in any one of these boxes, a coordinate i of this solution is randomly selected. This solution coordinate is then set equal to a random value in the range $[l_i^w, u_i^w]$. This procedure is repeated until the obtained solution no longer belongs to any of the boxes $\bar{B}^{w,j}$, $j = 1, \ldots, z$. Note that in this way, DEV works as optimizing a problem within a *cutting box*, where each new run on a certain level w generates a new cutting. Therefore, each one of the $maxSubBoxes$ runs of DEV on a given level w provides a different solution. In this way, local optima found in previous runs are discarded, increasing the possibility of the algorithm to find global optimal solutions. The best box found, i.e., the box associated to the best solution $\bar{x}^{w,z}$, is defined as box B^{w+1} on the next level of the tree. If the basic procedure fails to converge to a feasible solution on the z-th iteration of level w, the algorithm CBS advances directly to the next level considering only the boxes obtained in the previous iterations at level w. To test the convergence of the basic procedure, it is possible to use a tolerance factor α dependent on the size of B^w. For example, considering DEV as the basic procedure, we may consider the convergence tolerance $\epsilon = \alpha \max_{i=1,\ldots,n}\{u_i^w - l_i^w\}$. In cases where the ranges of the variables are disproportioned, it might be a good idea to adopt for DEV a different tolerance ϵ_i for each coordinate i, i.e., $\epsilon_i = \alpha(u_i^w - l_i^w)$.

The CBS algorithm descends in the search tree until level $maxLevels$ and then returns the best solution found. Algorithm 4 shows the CBS algorithm while Figure 1 illustrates a small example of a CBS tree.

4 Computational Results

In this section, we describe the computational results obtained with the application of the CBS framework combined with a metaheuristic to solve instances of the general nonlinear optimization problem. As previously mentioned, we adopt the DEV algorithm as the basic procedure on the application of CBS because it provides a mechanism for detecting convergence, required by CBS, and also because of its computation effectiveness, which is demonstrated in [17]. In [17], the authors report the successful application of DEV on a set of 22 test instances, which are described in [14]. In this paper, we consider a set of 36 instances with multiple local optima taken from CEC2010 (detailed information on the instances can be found in [16]). This set of test problems was constructed with 18 scalable instances, where each instance was included in the set with 10 and 30 variables, according to

[16]. We note, however, that the optimal solution of these test instances is not available in the literature. As recommended in [16], we adopt a tolerance of 10^{-4} with respect to the equality constraints (3).

The experiments were conducted with our own implementations of the algorithms in C++, compiled with g++ compiler (GCC 4.4) using the flags "-O3-march = native". Tests were run on a computer with Intel Core 2 Duo frequency 2.13 GHz and 2 MB cache, 2 GB of RAM and using the Linux operating system Ubuntu 11 (32 bits). For the generation of random numbers, we used the implementation of the Mersenne Twister method available in [8]
.

We compare the application of DEV itself with the application of DEV within the CBS framework, performing 100 runs on all test instances. In the first case, the parameter ϵ was set equal to $1.0e-6$, while in the second case, this parameter was defined as a function of the parameter α, as explained in Section 3. The other DEV parameters were adjusted for both cases as follows: $MAXGEN = 2000$, $P = 50$, $M = 4$, $CR^0 = 0.5$ and $Sr^0 = 1.0$ (see Algorithm 2 for a description of these parameters). The values used for the CBS parameters were $maxLevels = 2$, $maxSubBoxes = 3$, $\lambda = 0.1$ and $\alpha = 0.001$ (see Algorithm 4 for a description of these parameters). In the last level of the CBS tree, we adopted $maxSubBoxes = 1$. The values used for the parameters were determined empirically.

Tables 1 and 2 show the computational results for the 18 test instances with 10 and 30 variables, respectively. The first column (Problem) identifies the problem, while the second one (Alg) discriminates the approach. Considering 100 executions of each approach for each test instance, the remaining columns bring the value of the objective function in the best solution found (Best), the average value of the objective function (Avg), the objective function value of the worst solution found (Worse), the standard deviation (SD), the number of runs in which at least one feasible solution was found (Feas Sols), the average running time in seconds (Avg time (s)), the average number of the objective function evaluations (Avg Avals-F) and the average number of the constraint functions (2) and (3) evaluations (Avg C-Avals). The last lines in each table show the average values (considering the previous lines) for the approaches.

Regarding the instances with 10 variables (Table 1), we note that CBS shows better results than DEV for 8 out off the 18 instances, considering the average value of the objective function (C01, C02, C03, C08, C09, C10, C15 and C17). For the other instances, the performance of both approaches can be considered equivalent with respect to this criterion. CBS also shows the best results with respect to the worst solution for 3 instances (C01, C02

Table 1: Computational results for DEV and CBS + DEV ($n = 10$).

Problem	**Alg**	**Best**	**Avg**	**Worse**	**SD**	**Feas Sols**	**Avg Time(s)**	**Avg F-Avals**	**Avg C-Avals**
C01	DEV	-0.7473	-0.7465	-0.7259	3.87E-03	100	0.50	140,086	196,712
$n = 10$	CBS + DEV	-0.7473	-0.7473	-0.7473	1.26E-08	100	1.54	418,399	608,236
C02	DEV	-2.2777	-2.2681	-2.2308	1.08E-02	99	0.83	207,902	321,604
$n = 10$	CBS + DEV	-2.2777	-2.2775	-2.2717	1.12E-03	100	2.05	452,558	833,092
C03	DEV	5.80E-14	8.69804	8.87555	1.25E+00	100	0.09	98,441	158,054
$n = 10$	CBS + DEV	7.07E-14	8.52065	8.87555	1.75E+00	100	0.51	489,934	855,472
C04	DEV	-9.92E-06	-9.85E-06	-9.77E-06	2.91E-08	100	0.27	75,252	134,244
$n = 10$	CBS + DEV	-9.93E-06	-9.86E-06	-9.78E-06	2.78E-08	100	1.17	321,969	605,320
C05	DEV	-483.611	-483.611	-483.611	2.74E-08	100	0.41	79,040	141,494
$n = 10$	CBS + DEV	-483.611	-483.611	-483.611	2.25E-08	100	1.91	358,105	675,874
C06	DEV	-578.662	-578.662	-578.657	7.20E-04	100	2.02	258,315	392,142
$n = 10$	CBS + DEV	-578.662	-578.662	-578.661	2.02E-04	100	5.52	644,671	1,125,880
C07	DEV	3.27E-14	9.41E-14	2.56E-13	4.27E-14	100	0.26	117,171	117,976
$n = 10$	CBS + DEV	1.98E-14	8.82E-14	2.87E-13	4.25E-14	100	0.46	202,536	204,978
C08	DEV	2.38E-14	2.4498	10.9415	4.45E+00	100	0.54	103,038	124,544
$n = 10$	CBS + DEV	1.48E-14	0.3607	10.9415	1.65E+00	100	1.21	216,461	289,460
C09	DEV	2.44E-14	2.0718	4.4082	2.21E+00	100	0.26	80,040	126,700
$n = 10$	CBS + DEV	2.04E-14	1.0140	4.4082	1.86E+00	100	0.82	231,586	409,480
C10	DEV	5.30E-14	38.8051	41.7260	1.07E+01	100	0.86	126,257	208,152
$n = 10$	CBS + DEV	2.77E-14	37.1361	41.7262	1.31E+01	100	2.89	393,998	715,520
C11	DEV	-	-	-	0.00E+00	0	1.95	400,050	400,050
$n = 10$	CBS + DEV	-	-	-	0.00E+00	0	1.95	400,050	400,050
C12	DEV	-	-	-	0.00E+00	0	1.10	215,085	400,050
$n = 10$	CBS + DEV	-	-	-	0.00E+00	0	1.09	215,380	400,050
C13	DEV	-68.4294	-68.4294	-68.4294	5.82E-07	100	1.66	254,853	327,326
$n = 10$	CBS + DEV	-68.4294	-68.4294	-68.4294	3.65E-07	100	3.42	484,093	707,938
C14	DEV	2.05E-14	8.63E-14	3.83E-13	4.87E-14	100	0.54	113,913	121,352
$n = 10$	CBS + DEV	2.17E-14	8.91E-14	6.66E-13	7.25E-14	100	0.97	187,303	230,278
C15	DEV	2.27E-14	3.2692	3.6732	1.16E+00	100	0.96	103,807	153,988
$n = 10$	CBS + DEV	4.31E-14	2.8225	3.6732	1.50E+00	100	2.39	235,291	392,044
C16	DEV	0	1.54E-15	1.31E-14	2.52E-15	100	0.34	37,815	66,390
$n = 10$	CBS + DEV	0	6.89E-16	5.90E-15	1.41E-15	100	2.14	227,495	424,288
C17	DEV	1.67E-14	0.0322	2.9011	3.06E-01	90	0.30	83,931	148,926
$n = 10$	CBS + DEV	1.17E-14	2.76E-09	2.51E-07	2.63E-08	91	1.21	328,482	615,150

Table 1: Computational results for DEV and CBS + DEV (*continuation of the last page*).

Problem	**Alg**	**Best**	**Avg**	**Worse**	**SD**	**Feas Sols**	**Avg Time(s)**	**Avg F-Avals**	**Avg C-Avals**
C18	DEV	1.22E-25	1.73E-18	1.14E-16	1.17E-17	100	0.53	137,999	170,662
$n = 10$	CBS + DEV	4.91E-27	4.18E-18	3.24E-16	3.25E-17	100	1.68	359,987	573,032
AVG	DEV	-188.9546	-98.0355	-96.4662	1.83E+00	88	0.74	146277	206131
$n = 10$	CBS + DEV	-188.9546	-108.3873	-106.4096	2.21E+00	88	1.83	342683	525898

Table 2: Computational Results for DEV and CBS + DEV ($n = 30$).

Problem	**Alg**	**Best**	**Avg**	**Worse**	**SD**	**Feas Sols**	**Avg Time(s)**	**Avg F-Avals**	**Avg C-Avals**
C01	DEV	-0.8219	-0.8170	-0.7970	5.97E-03	100	1.78	198,204	248,524
$n = 30$	CBS + DEV	-0.8219	-0.8213	-0.8179	1.25E-03	100	5.52	583,333	787,746
C02	DEV	-2.1787	-1.8375	-1.2136	2.49E-01	100	2.66	242,334	382,998
$n = 30$	CBS + DEV	-2.2258	-2.1466	-1.0326	1.94E-01	100	7.24	606,947	1,081,060
C03	DEV	28.6735	28.6735	28.6735	2.86E-06	100	0.24	115,227	199,274
$n = 30$	CBS + DEV	28.6735	28.6735	28.6735	1.82E-06	100	0.88	401,346	740,778
C04	DEV	-	-	-	0.00E+00	0	2.05	208,419	400,050
$n = 30$	CBS + DEV	-	-	-	0.00E+00	0	2.05	208,419	400,050
C05	DEV	-152.873	-68.192	81.053	4.03E+01	84	3.22	236,669	398,830
$n = 30$	CBS + DEV	-231.581	-153.885	-88.324	3.18E+01	82	8.43	579,475	1,069,030
C06	DEV	-513.581	-268.165	-91.7801	1.29E+02	99	6.00	252,325	398,278
$n = 30$	CBS + DEV	-528.739	-392.73	-194.225	1.16E+02	99	15.38	590,340	1,082,330
C07	DEV	6.7740	15.1399	72.6330	1.42E+01	100	1.31	251,400	254,402
$n = 30$	CBS + DEV	5.4182	9.7708	15.4738	2.09E+00	100	2.54	478,792	487,884
C08	DEV	10.0434	21.076	118.686	2.17E+01	100	3.29	223,933	255,610
$n = 30$	CBS + DEV	6.75828	10.904	15.8365	1.82E+00	100	7.50	468,721	597,180
C09	DEV	481.97	4,810.52	26,692.00	5.11E+03	100	2.06	255,335	368,486
$n = 30$	CBS + DEV	97.81	268.36	2,937.10	3.53E+02	100	5.27	562,196	984,846
C10	DEV	251.37	2,739.53	25,440.50	3.22E+03	100	4.73	258,340	369,554
$n = 30$	CBS + DEV	53.63	180.94	1,771.81	2.87E+02	100	12.01	567,108	979,536
C11	DEV	-	-	-	0.00E+00	0	5.90	400,050	400,050
$n = 30$	CBS + DEV	-	-	-	0.00E+00	0	6.00	400,050	400,050
C12	DEV	-	-	-	0.00E+00	0	3.04	208,409	400,050

Table 2: Computational Results for DEV and CBS + DEV (*continuation of the last page*).

Problem	**Alg**	**Best**	**Avg**	**Worse**	**SD**	**Feas Sols**	**Avg Time(s)**	**Avg F-Avals**	**Avg C-Avals**
$n = 30$	CBS + DEV	-	-	-	0.00E+00	0	3.04	208,455	400,050
C13	DEV	-66.9411	-63.1009	-57.0551	1.88E+00	100	5.76	310,418	400,050
$n = 30$	CBS + DEV	-67.4158	-64.7474	-62.8394	1.04E+00	100	13.40	658,411	985,986
C14	DEV	11.9862	18.635	75.201	1.16E+01	100	3.14	223,475	254,376
$n = 30$	CBS + DEV	8.18846	22.0326	85.6042	2.44E+01	100	6.80	441,256	577,406
C15	DEV	21.9435	59.0175	454.892	6.22E+01	100	4.87	183,081	265,536
$n = 30$	CBS + DEV	21.6046	21.728	27.3195	8.03E-01	100	14.51	491,955	837,683
C16	DEV	7.29E-14	0.0001	0.0132982	1.33E-03	100	3.13	118,935	211,110
$n = 30$	CBS + DEV	1.68E-15	1.17E-14	2.80E-14	4.78E-15	100	10.85	395,967	742,782
C17	DEV	0.0706	0.5871	3.1105	7.09E-01	100	1.87	216,135	324,316
$n = 30$	CBS + DEV	0.0127	0.0626	1.0916	1.11E-01	100	5.21	532,203	948,396
C18	DEV	3.7411	19.2012	77.9627	1.27E+01	100	2.92	240,601	341,366
$n = 30$	CBS + DEV	0.0360	1.9193	6.5721	1.44E+00	100	7.36	531,697	920,918
AVG	DEV	5.7265	487.3512	3526.2586	5.08E+02	82	3.2202	230,183	326,270
$n = 30$	CBS + DEV	-43.4749	-4.9962	324.4459	51.2059	82	7.4442	483,704	719,419

and C17). Regarding the best solution, both approaches have equivalent performence on all test instances. We note that, considering these three criteria, CBS does not underperform DEV on any instance. For instances C11 and C12, no approach can find any feasible solution. We point out that, since CBS uses DEV as the basic procedure in this study, in cases where DEV fails to find a feasible solution, CBS also fails. As would be expected, the computational effort demanded by CBS was greater than that demanded by DEV. On average, the computational time for CBS is 2.45 times the time for DEV. We emphasize that each run of DEV within the CBS framework tends to be computationally cheaper than an independent run of DEV due to difference in the value of the convergence tolerance ϵ that was used. The number of evaluations of the objective function and the constraints performed by CBS are, on average, 2.34 and 2.55 times higher than the ones performed by DEV, respectively.

Regarding the instances with 30 variables (Table 2), we note that CBS shows better results than DEV for 12 out off the 18 instances (all except C1, C3, C4, C11, C12 and C16), considering the best obtained solution. For other instances, both approaches show equivalent performance with respect to this criterion. Considering the average value of the objective function, CBS shows better performance than DEV for 13 instances (all, except C3, C4, C11, C12 and C14). Only for instance C14, DEV shows better result than CBS, considering the average value of the objective function. This same behavior is repeated for the worst solution, where CBS gives better results than DEV for 12 instances (all except C02, C03, C04, C11, C12, and C14). With respect to the computational time and the average number of evaluations of the objective and constraints functions, the results for CBS are on average 2.31, 2.10 and 2.2 higher than those for DEV, respectively. Finally, we point out that both approaches cannot provide any feasible solution for instances C04, C11 and C12.

The computational results presented in this section, especially those on Table 2 demonstrate the effectiveness of the CBS algorithm. It is clear from the results that the application of CBS can improve the performance of the metaheuristic adopted to solve the problem addressed here, although this metaheuristic is applied multiple times on the same problem. The main reasons for the success of CBS are to consider boxes that gradually become smaller through the levels of the CBS tree, and its ability to prevent local optima already found to be revisited in future applications of the basic procedure on the same level of CBS tree.

5 Conclusion

Continuous global optimization problems are in general difficult to adress with classical solution methods. Although in recent years there has been a major effort in developing good metaheuristics that attempt to reach the global optimal solution of these problems, in many cases these algorithms still fail to accomplish this goal, especially when the problem has many local optima.

In this paper, we propose a new framework to improve the performance of metaheuristics for the general nonlinear optimization problem called Cutting Box Strategy (CBS). CBS uses a basic procedure that is successively applied in an optimization process inside a box, where the boxes are progressively reduced. CBS can be considered as a search tree algorithm, where the search spaces are boxes. On each level of the tree the basic procedure is applied several times to the problem, considering only smaller boxes that are contained in the original one. The different aspect of CBS is that it relies on an intelligent application of multi-starts to the subproblems on each level of the tree, which avoids the same solution to be generated more than once, by creating a "hole" in the box where the search for solutions is performed, and consequently preventing the basic procedure to generate solutions in these "holes".

The methodology described here is general and therefore can be applied to any stochastic basic procedure or metaheuristic with a mechanism for detecting convergence that can handle the addressed problem. We evaluated the performance of CBS combined with a good metaheuristic on a set of 36 test instances with multiple local optima. The results showed that CBS was able to considerably improve the results of the metaheuristic adopted as the basic procedure considering the quality of the solutions obtained in exchange for a tolerable increase in computational effort.

As possible future research, we suggest the evaluation of the performance of CBS with other basic procedures and test instances, along with the development of other strategies for exploiting the CBS tree beyond the search procedure used here.

References

[1] Shamim Akhtar, Kang Tai, and Tapabrata Ray. A socio-behavioural simulation model for engineering design optimization. In *Projects in India, by Ruth Alsop with Ved*, pages 200–2, 2002.

[2] M.M. Ali and Z. Kajee-Bagdadi. A local exploration-based differential evolution algorithm for constrained global optimization. *Applied Mathematics and Computation*, 208(1):31 – 48, 2009.

[3] B.V. Babu and Rakesh Angira. Modified differential evolution (mde) for optimization of non-linear chemical processes. *Computers & Chemical Engineering*, 30:989 – 1002, 2006.

[4] S. Ben Hamida and M. Schoenauer. Aschea: new results using adaptive segregational constraint handling. In *Evolutionary Computation, 2002. CEC '02. Proceedings of the 2002 Congress on*, volume 1, pages 884 –889, may 2002.

[5] Uday K. Chakraborty. *Advances in Differential Evolution.* Springer Publishing Company, Incorporated, 1 edition, 2008.

[6] S. Das and P.N. Suganthan. Differential evolution: A survey of the state-of-the-art. *Evolutionary Computation, IEEE Transactions on*, 15(1):4 –31, feb. 2011.

[7] Kalyanmoy Deb. An efficient constraint handling method for genetic algorithms. *Computer Methods in Applied Mechanics and Engineering*, 186:311 – 338, 2000.

[8] A. Fog. Pseudo random number generators uniform and non-uniform distributions. http://www.agner.org/random/ accessed 7-July-2012.

[9] Roger Gämperle, Sibylle D. Müller, and Petros Koumoutsakos. A parameter study for differential evolution. In *WSEAS Int. Conf. on Advances in Intelligent Systems, Fuzzy Systems, Evolutionary Computation*, pages 293–298. Press, 2002.

[10] Abdel-Rahman Hedar and Masao Fukushima. Derivative-free filter simulated annealing method for constrained continuous global optimization. *J. of Global Optimization*, 35(4):521–549, August 2006.

[11] F. Huang, Ling Wang, and Qie He.

[12] Slawomir Koziel and Zbigniew Michalewicz. Evolutionary algorithms, homomorphous mappings, and constrained parameter optimization. *Evol. Comput.*, 7(1):19–44, March 1999.

[13] J. Lampinen. A constraint handling approach for the differential evolution algorithm. In *Evolutionary Computation, 2002. CEC '02. Proceedings of the 2002 Congress on*, volume 2, pages 1468 –1473, 2002.

[14] J.J. Liang, T.P. Runarsson, E. Mezura-Montes, M. Clerc, P.N. Suganthan, C.A. Coello Coello, and K. Deb. Problem definitions and evaluation criteria for the cec2006 special session on constrained real-parameter optimization. In: Report, 2006.

[15] Hui Liu, Zixing Cai, and Yong Wang. Hybridizing particle swarm optimization with differential evolution for constrained numerical and engineering optimization. *Applied Soft Computing*, 10(2):629 – 640, 2010.

[16] R. Mallipeddi and P. N. Suganthan. Problem definitions and evaluation criteria for the cec 2010 competition on constrained real- parameter optimization. In: Report, 2010.

[17] Wendel Melo, Marcia Fampa, and Fernanda Raupp. Differential evolution variant for constrained global continuous optimization. Submitted to *Computers & Industrial Engineering*, 2012.

[18] Wendel Melo, Marcia Fampa, and Fernanda Raupp. A stochastic local search algorithm for constrained continuous global optimization. *International Transactions in Operational Research*, 2012.

[19] E. Mezura-Montes and C.A.C. Coello. A simple multimembered evolution strategy to solve constrained optimization problems. *Evolutionary Computation, IEEE Transactions on*, 9(1):1 – 17, feb. 2005.

[20] Efén Mezura-Montes and Carlos A. Coello Coello. Constraint-handling in nature-inspired numerical optimization: Past, present and future. *Swarm and Evolutionary Computation*, 1(4):173 – 194, 2011.

[21] Efrén Mezura-Montes, Carlos A. Coello Coello, Edy I. Tun-Morales, Sección De Computación, and Villahermosa Tabasco. Simple feasibility rules and differential evolution for constrained optimization. In *In Proceedings of the 3rd Mexican International Conference on Artificial Intelligence (MICAI 2004*, pages 707–716. Springer Verlag, 2004.

[22] Efrén Mezura-Montes, Carlos A. Coello Coello, and Jesús Velázquez-Reyes. Increasing successful offspring and diversity in differential evolution for engineering design. In *Proceedings of the Seventh International Conference on Adaptive Computing in Design and Manufacture*, ACDM 2006, pages 131–139, 2006.

[23] Efrén Mezura-Montes, Jesús Velázquez-Reyes, and Carlos A. Coello Coello. Promising infeasibility and multiple offspring incorpo-

rated to differential evolution for constrained optimization. In *Proceedings of the 2005 conference on Genetic and evolutionary computation*, GECCO '05, pages 225–232, New York, NY, USA, 2005. ACM.

[24] Ferrante Neri and Ville Tirronen. Recent advances in differential evolution: a survey and experimental analysis. *Artificial Intelligence Review*, 33:61–106, 2010. 10.1007/s10462-009-9137-2.

[25] Chandra Sekhar Pedamallu and Linet Ozdamar. Investigating a hybrid simulated annealing and local search algorithm for constrained optimization. *European Journal of Operational Research*, 185(3):1230 – 1245, 2008.

[26] Kenneth Price, Rainer M. Storn, and Jouni A. Lampinen. *Differential Evolution: A Practical Approach to Global Optimization (Natural Computing Series)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[27] A.K. Qin, V.L. Huang, and P.N. Suganthan. Differential evolution algorithm with strategy adaptation for global numerical optimization. *Evolutionary Computation, IEEE Transactions on*, 13(2):398 –417, april 2009.

[28] T. Ray and K.M. Liew. Society and civilization: An optimization algorithm based on the simulation of social behavior. *Evolutionary Computation, IEEE Transactions on*, 7(4):386 – 396, aug. 2003.

[29] T.P. Runarsson and Xin Yao. Search biases in constrained evolutionary optimization. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 35(2):233 –243, may 2005.

[30] Rainer Storn and Kenneth Price. Differential evolution a simple and efficient heuristic for global optimization over continuous spaces. *J. of Global Optimization*, 11(4):341–359, December 1997.

[31] Zhenyu Yang, Xin Yao, and Jingsong He. Making a difference to differential evolution. In Patrick Siarry and Zbigniew Michalewicz, editors, *Advances in Metaheuristics for Hard Optimization*, Natural Computing Series, pages 397–414. Springer Berlin Heidelberg, 2007.

[32] Min Zhang, Wenjian Luo, and Xufa Wang. Differential evolution with dynamic stochastic selection for constrained optimization. *Information Sciences*, 178(15):3043 – 3074, 2008. <ce:title>Nature Inspired Problem-Solving</ce:title>.

[33] Karin Zielinski and Rainer Laur. Stopping criteria for differential evolution in constrained single-objective optimization. In Uday Chakraborty, editor, *Advances in Differential Evolution*, volume 143 of *Studies in Computational Intelligence*, pages 111–138. Springer Berlin / Heidelberg, 2008. 10.1007/978-3-540-68830-3_4.