Boundary Search for Constrained Numerical Optimization Problems

Guillermo Leguizamón and Carlos Coello Coello

Abstract The necessity of approaching the boundary between the feasible and infeasible search space for many constrained optimization problems is a paramount challenge for every constraint-handling technique. It is true that many of the state-of-the-art constraint-handling techniques performs well when facing constrained problems. However, it is a common situation that reaching the boundary between the feasible and infeasible search space could be a difficult task for some particular problems. Firstly, this chapter shows a general overview of the constraint-handling techniques based on a boundary approach and emphasizing a recent proposal applying a more general boundary operator. In addition, the chapter includes some particular considerations related to the implementation aspects of the boundary approach when facing problems with one o more constraints. Another important issue also considered here is about the implementation of this approach when taking into account different search engines. On this direction, some basic examples are depicted as guidelines for possible implementations under well-known metaheuristics as Evolutionary Algorithms (EAs), Particle Swarm Optimization (PSO), and Ant Colony Optimization (ACO). To validate the boundary approach implemented under the above metaheuristics, an experimental study is presented in which well-known problems were considered. Finally, a brief summary of the chapter and some ideas for future works are given which could help the researchers interested in developing advanced constraint-handling techniques.

Key words: Constrained Handling, Boundary Search, Particle Swarm Optimization, Ant Colony Optimization

Guillermo Leguizamón
LIDIC - Universidad Nacional de San Luis Ejército de Los Andes 950 San Luis 5700, ARGENTINA e-mail: legui@unsl.edu.ar

Carlos A. Coello Coello
CINVESTAV-IPN Evolutionary Computation Group (EVOCINV) Departamento de Computación Av. IPN No. 2508. Col. San Pedro Zacatenco México D.F. 07300, MÉXICO e-mail: ccoello@cs.cinvestav.mx

1 Introduction

The boundary search can be considered as alternative approach when facing problems with active constraints (see Sect. 2) at the optimal or high quality solutions. This is mainly observed for that problems that include at least one equality constraint. However, there exist plenty of optimization problems without any equality constraint for which many of their constraints are active for the best feasible solutions. Clearly the more appropriate situation for the boundary approach is when the problem has only one equality constraint. In addition, the boundary search could be used as a complementary mechanism of another constraint-handling technique to rapidly reach or force the exploration towards the regions around the boundary between the feasible and infeasible search space.

Specific operators (or *ad hoc* boundary operators) could be the right candidates to search only on the boundary region between the feasible and infeasible search space. However, it is not always possible to design specific boundary operators for each problem constraint. Furthermore, there exist only a few examples of this kind of boundary operators in the literature. Michalewicz et al. [17] wrote one of the first papers on boundary search through the use of evolutionary algorithms for constrained numerical optimization problems. The efficiency of this approach was shown by using two constrained optimization problems: Keane's function (also known as *G*02) [10] and another function with one equality constraint (also known as *G*03). For solving these problems the authors proposed two genetic operators which generate offspring lying on the boundary between the feasible and infeasible search space. Similarly, Schoenauer and Michalewicz [19] proposed several evolutionary operators capable of exploring a general surface of dimension $n-1$ (n is the number of variables). The design of these operators, tested on three problems, depends on the surface representation: curves-based, plane-based, and parametric representation. Although not using an *ad hoc* boundary operator, Wu et al. [23] proposed a GA for the optimization of a water distribution system, which is a highly constrained optimization problem. The proposed approach co-evolves and self-adapts two penalty factors in order to guide and preserve the search towards the boundary of the feasible search space.

On the other hand, Gottlieb [9] introduces and remarks the use of the the boundary approach for a combinatorial optimization problem, more precisely, the multiple knapsack problem. By the same year Leguizamón and Michalewicz proposed for the same problem, an Ant System which biases the search boundary region and gives encouraging results [15]. The maximum independent set problem is also considered under the same approach and plenty instances of this problem were solved optimally [7].

The reduction of the search space is one of the most relevant characteristics of the boundary search approach since the exploration considers only the boundary of the feasible search space. However, many of the test cases considered in the former works only include problems with one constraint for which it is possible to define *ad hoc* genetic operators that fit perfectly the boundary of the feasible region. However, this sort of approach is impractical in an arbitrary problem with many con-

straints, and it is therefore necessary to define a more general approach for boundary search which can be as robust as possible to deal with different types of constraints. More recently, Leguizamón and Coello [13, 14] proposed a boundary approach that focuses the search on the boundary region by considering a sort of more general boundary operator applicable to any type of constraints. The experimental reports show the applicability of the boundary approach using ant colony based algorithms as a search engine.

The next section of this chapter shows a general overview of the constraint-handling techniques. Sect. 3 describes the boundary search approach and two alternatives for exploring the boundary between the feasible and infeasible search space: *ad hoc* operators and a general operator. In the last case, emphasizing a more recent proposal according to Leguizamón and Coello [13, 14]. In order to visualize some considerations about specific implementation aspects of the boundary approach, Sect. 4 displays the pseudocode by taking into account different search engines. On this direction, some basic examples are depicted as guidelines for possible implementations under well-known metaheuristics (MHs) as Evolutionary Algorithms [1, 6, 8], Particle Swarm Optimization [6, 11, 12], and Ant Colony Optimization [4–6] which can be have been used as a search engine to successfully implement constraint-handling techniques. Section 5 shows the results of the application of EAs, ACO, and PSO to a well-known testbed of numerical optimization problems. Finally, a brief summary and some ideas for future work are given which could help the researchers interested in developing advanced constraint-handling techniques.

2 A General Overview of Constraint-Handling Techniques

The general nonlinear programming problem whose aim is to find $\mathbf{x}$ so as to optimize:

$$f(\mathbf{x}) \quad \mathbf{x} = (x_1, x_2, ..., x_n) \in \mathbb{R}^n$$

where $\mathbf{x} \in \mathscr{F} \subset \mathscr{S}$. The set $\mathscr{S} \subset \mathbb{R}^n$ defines the search space and sets $\mathscr{F} \subset \mathscr{S}$ and $\mathscr{U} = \mathscr{S} - \mathscr{F}$ define the *feasible* and *infeasible* search spaces, respectively. The search space $\mathscr{S}$ is defined as an n-dimensional rectangle in $\mathbb{R}^n$ (domains of variables defined by their lower and upper bounds):

$$l(i) \leq x_i \leq u(i) \text{ for } 1 \leq i \leq n$$

whereas the feasible set $\mathscr{F}$ is defined by the intersection of $\mathscr{S}$ and a set of additional $m \geq 0$ constraints:

$$g_j \leq 0, \text{ for } j = 1, \ldots, q \text{ and } h_j = 0 \text{ for } j = q+1, \ldots, m.$$

At any point $\mathbf{x} \in \mathscr{F}$, the constraints g_k that satisfy $g_k(\mathbf{x}) = 0$ are called the active constraints at $\mathbf{x}$. Equality constraints h_j are active at all points of $\mathscr{F}$.

The most common way of incorporating constraints into MHs (e.g., EAs, DE, PSO) have been penalty functions which are the oldest approach used as a constraint-handling technique. However, due to the well-known difficulties associated with them, researchers in MHs (mainly in EAs) have proposed different ways to automate the definition of good penalty factors, which remains as the main drawback of using penalty functions. Additionally, several researchers have developed a considerable amount of alternative approaches to handle constraints, mainly to deal with specific features of some complex optimization problems in which it is difficult to estimate good penalty factors or to even generate a single feasible solution.

A comprehensive survey of constraint-handling techniques that have been adopted over the years to handle all sorts of constraints (linear, non-linear, equality, and inequality) in EAs can be found in Coello [3]. This survey covers extensively: a) penalty functions in several of their variations that have been used with EAs (i.e., static, dynamic, annealing, adaptive, co-evolutionary, and death penalties); b) the use of special representations and genetic operators (e.g., operators that preserve feasibility at all times and decoders that transform the shape of the search space); c) repair algorithms, which are normally used in combinatorial optimization problems in which the traditional genetic operators tend to generate infeasible solutions all (or at least most of) the time. Thus, ”repair” refers, in this context, to make valid (or feasible) these individuals through the application of a certain (normally heuristic) procedure; d) techniques that handle objectives and constraints separately; and e) discusses approaches that use hybrids with other techniques such as Lagrangian multipliers or fuzzy logic as well as other more novel approaches.

Although the Coello's survey is mainly concerned with constraint-handling techniques from the perspective of EAs, the concepts depicted conform a general framework to be applied with other search engines. Examples of the more recent applications of using novels MHs (e.g., DE, PSO, ACO, etc.) for constraint handling techniques can be found at the web site from EVOCINV [2] which includes upto date references to the more representatives constraint-handling techniques implemented under different search engines.

It is worth remarking that plenty of problems formulated as at the beginning of this section, include active constraints at the best known or optimal solutions. For example, for problems with at least one equality constraint h_j, the respective optimal solution will lay on the region defined by $h_j(\mathbf{x}) = 0$. Furthermore, for many problems, the best solutions may lay on the boundary between the feasible and infeasible search space of some inequality constrains, i.e., the region defined by $g_j(\mathbf{x}) = 0$. When those conditions are met for a particular problem, the design of *ad hoc* operators or approaches that explore the search space focusing on the boundary region (according either to the equality and/or inequality constraints) can be a suitable alternative for including in a specific search engine or metaheuristic.

3 The Boundary Search Approach

In the following we first explain how the boundary region can be approached given a specific search space; more precisely, a subset of the n-dimensional space $\mathbb{R}^n$. Then, we also describe the manner in which this search space could be explored assuming a hypothetical search engine and exploration operators, as well as the properties that they should satisfy. Afterwards, we present in detail the proposed technique that takes advantage of the boundary approach to explore some specific regions of the boundary of the feasible search space by considering *ad hoc* and a general boundary operators.

Definition 1. Given a constrained numerical optimization problem, its determines a feasible search space $\mathscr{F}$. In addition, the problem constraints determine a set $\mathscr{F}_{\mathscr{B}} \subseteq \mathscr{F}$ which represent the points in $\mathscr{F}$ which make active at least one of the problem constraints ($\mathscr{F}_{\mathscr{B}} = \mathscr{F}$ when the problem includes only one equality constraint).

To appropriately define boundary operators , we must take into account that set $\mathscr{F}_{\mathscr{B}}$ must be closed under the application of a boundary operator. Let us suppose that Ω_r is a r-ary boundary operator, i.e., it takes r points in set $\mathscr{F}_{\mathscr{B}}$; then the resulting point must be in $\mathscr{F}_{\mathscr{B}}$. In other words, an r-ary boundary operator can generally be defined as $\Omega_r : (\mathscr{F}_{\mathscr{B}})^r \rightarrow \mathscr{F}_{\mathscr{B}}$. For example, when considering a "boundary" *crossover operator* that takes two parents to generate one child, it can be defined in the boundary context as $\Omega_2 : \mathscr{F}_{\mathscr{B}} \times \mathscr{F}_{\mathscr{B}} \rightarrow \mathscr{F}_{\mathscr{B}}$. Fig. 1 display a set of points laying on the boundary region with respect to a problem constraint and the application of boundary operators that take respectively 2, 3, and 4 point as argument. Clearly, Ω_r operator could be any operator (e.g., genetic operators as in EAs) or equation (e.g., velocity and position updating of the particles in PSO) used in different MHs used for sampling new points in the search space.

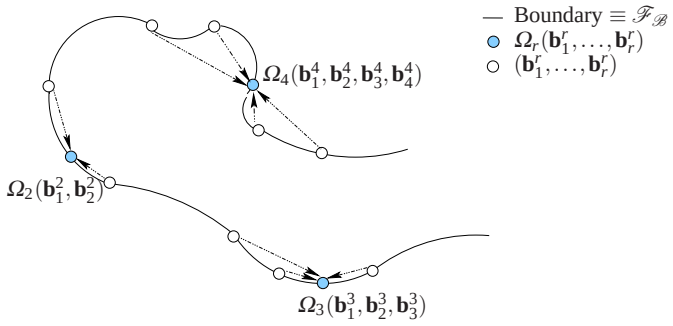

Fig. 1: This figure shows a set of points laying on $\mathscr{F}_{\mathscr{B}}$ (white circles) and the respective points sampled on $\mathscr{F}_{\mathscr{B}}$ (filled circles) after the application of the boundary operator Ω_r, for $r = 2, 3, 4$. Notice that $\mathscr{F}_{\mathscr{B}}$ is closed under the application of Ω_r

It is worth noting that the definition of an Ω_r operator which makes set $\mathscr{F}_{\mathscr{B}}$ closed under its application could be a difficult or at least impossible task for most of the usual problem constraints. The most paradigmatic case is the proposal of Michalewicz et al. [17] where different boundary operators where defined which perfectly fit in our definition of the Ω_r operator: *Geometrical Crossover*, *Spherical Crossover*, and *ad- hoc* mutation operators. However, their application is limited to couple of problems with specific characteristics. In order to mitigate this drawback, the design of more general operators to explore $\mathscr{F}_{\mathscr{B}}$ could be an interesting approach when considering a wider set of problems to be tackled under the boundary approach. It must be noted that the above classification, i.e., *ad hoc* and general operators, is not intended to be a general rule, instead, it only represents the authors' point of view in the way the boundary approach could be conceived.

In the following we present two possibles alternative to define an Ω_r operator, either by *ad hoc* operators or a general operator. The main difference among these two types of operators is that *ad hoc* operators are defined to operate on the *phenotype space* where as the general operator does it on the *genotype space* (Sect. 3.2 describes these concepts in more detail).

3.1 Ad hoc ***boundary operators***

As mentioned in a previous section, the work of Michalewicz et al. [17] represents one of the first intents to define specific *ad hoc* operators to explore the boundary between the feasible and infeasible search space. Although this approach to explore the boundary region can be useful and efficient, it in not always possible to define a specific one for any problem constraint. Indeed, for most of the problem constraints, to find an adequate operator could be as difficult as solving the original problem. As a manner of showing the way in which the boundary region can be explored, we will describe in the following a classical example, the Keane's problem [10]. The reason for showing this alternative through an example is because an *ad hoc* operator completely depends on the shape of the involved constraints.

3.1.1 Exploration of the boundary region under *ad-hoc* operators

The exploration of the boundary search space under *ad hoc* operators can be visualized more clearly for a particular problem since its definition depends on the particular constraint considered. For our example, we have chosen a very well-known constraint optimization problem widely used as a benchmark to test different constraint-handling techniques: the Keane's problem. This problem, also known as *G*02 (see [16]) includes a non-linear objective function and two inequality constraints. More precisely, an optimal solution for *G*02 aims at maximizing:

$$f(\mathbf{x}) = |\frac{\sum_{i=1}^{n} cos^4(x_i) - 2\prod_{i=1}^{n} cos^2(x_i)}{\sqrt{\sum_{i=1}^{n} ix_i^2}}|$$

subject to:

$$g_1(\mathbf{x}) = 0.75 - \prod_{i=1}^{n} x_i \leq 0$$
$$g_2(\mathbf{x}) = \sum_{i=1}^{n} x_i - 7.5n \leq 0$$

where $n = 20$ and $\mathscr{S} = \{\mathbf{x} \in \mathbb{R}^{20} | 0 \leq x_i \leq 10, \text{ for } i = 1 \ldots 20\}$. The best known solution is at $\mathbf{x}^* = (3.16237443645701, 3.12819975856112, 3.09481384891456, 3.06140284777302, 3.02793443337239, 2.99385691314995, 2.95870651588255, 2.92182183591092, 0.49455118612682, 0.48849305858571, 0.48250798063845, 0.47695629293225, 0.47108462715587, 0.46594074852233, 0.46157984137635, 0.45721400967989, 0.45237696886802, 0.44805875597713, 0.44435772435707, 0.44019839654132)$ where $f(\mathbf{x}^*) = 0.80619$ and constraint g_1 is close to being active. Fig. 2 shows a plotting of the *G*02's objective function for $n = 2$.

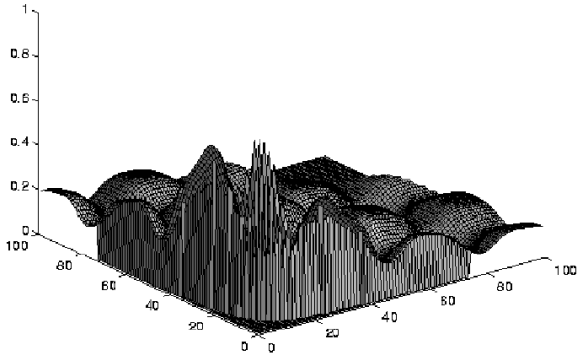

Fig. 2: Keane's function with $n = 2$. Infeasible solutions were assigned value zero

Despite of problem *G*02 has two constraints, one of them is just dismissed when solving this problem since: a) the second constraint is satisfied for all solutions laying on the boundary of the first constraint and b) the first constraint is close to being active at best known solution (see [17]). For this problem, the space $\mathscr{F}_{\mathscr{B}} = \{\mathbf{x} \in \mathscr{S} | 0.75 - \prod_{i=1}^{20} x_i = 0\}$

The search engine used in Michalewicz et al. [17] is an EA where the main components are described as follows:

i. **Initialization**: Randomly choose a positive variable for x_i and use it inverse as a variable for x_{i+1}. The last variable is either 0.75 (when n is odd) or multiplied by 0.75 (if n is even).
ii. **Mutation**: is a unary operator ($r = 1$) defined by

$$\Omega_1(\mathbf{x}) = (x_1, \ldots, q \times x_i, \ldots, \frac{1}{q} \times x_j, \ldots, x_n),$$

where q is a random factor restricted to respect the bounds on the variables and $1 \leq i, j \leq 20$ randomly chosen, with $i \neq j$.
iii. **Crossover**: is a binary operator ($r = 2$) called *geometrical crossover* and defined according to our notation by

$$\Omega_2(\mathbf{x}, \mathbf{y}) = (x_1^{\alpha} y_1^{1-\alpha}, \ldots, x_n^{\alpha} y_n^{1-\alpha}), \text{ with } 1 \leq \alpha \leq 1,$$

where α is a random number.

By using the above initialization procedure, all points in the initial population will lay on $\mathscr{F}_{\mathscr{B}}$. Similarly, $\mathscr{F}_{\mathscr{B}}$ is closed under the application of *ad hoc* operators Ω_1 and Ω_2, therefore, all points generated will also lay on the boundary. It is worth remarking that the application of boundary operators for problem $G02$ produced at least two very important results in the area of constraint-handling techniques. First of all, tgood quality solutions were formerly obtained by using a boundary operator and second, showed the usefulness and potential of the boundary approach for certain types of numerical optimization problems.

3.2 A general boundary operator

We describe here an alternative[1] general boundary approach (proposed in [13, 14]) which is based on the notion that each point $\mathbf{b}$ of the boundary region can be represented by means of two different points $\mathbf{x}$ and $\mathbf{y}$, where $\mathbf{x}$ is some feasible point and $\mathbf{y}$ is some infeasible one, i.e., $(\mathbf{x}, \mathbf{y})$ can represent one point lying on the boundary by applying a "binary search" on the straight line connecting the points $\mathbf{x}$ and $\mathbf{y}$ (when considering an equality constraint, $\mathbf{z} \in \mathscr{F}$ iff $h(\mathbf{z}) \leq 0$; otherwise, $\mathbf{z} \in \mathscr{U}$). Fig. 3 shows a hypothetical search space including the feasible (shadowed area) and infeasible regions. We can identify four points lying on the boundary $\mathbf{b}_1$, $\mathbf{b}_2$, $\mathbf{b}_3$, and $\mathbf{b}_4$ which are respectively obtained from $(\mathbf{x}_1, \mathbf{y}_1)$, $(\mathbf{x}_2, \mathbf{y}_2)$, $(\mathbf{x}_3, \mathbf{y}_3)$, and $(\mathbf{x}_4, \mathbf{y}_4)$.

The binary search applied to each pair of points $(\mathbf{x}, \mathbf{y})$ is achieved following the steps described in function BS (see Algorithm 1). For example, a possible application of this process can be seen in Fig. 3 where we adopt the pair of points $(\mathbf{x}_3, \mathbf{y}_3)$ from which we obtain the point $\mathbf{b}_3$, which lies on the boundary. The first step (labeled (1)) indicates that the first mid point found is feasible. Consequently, the left

[1] It is possible that other general operators can be visualized to implement under the boundary search approach.

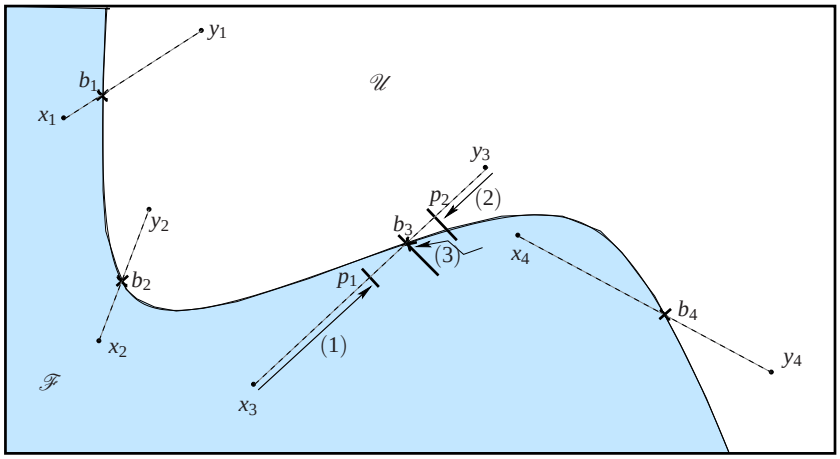

Fig. 3: Given one feasible and one infeasible point, the respective point lying on the boundary can be easily reached by using a simple binary search. In this way, the each point on the boundary can be reached from at least a pair of points $(\mathbf{x},\mathbf{y})$ with $\mathbf{x} \in \mathscr{F}$ and $\mathbf{y} \in \mathscr{U}$

side of the straight line ($\mathbf{x}_3$) is moved to point $\mathbf{p}_1$. In the next step (labeled (2)) we consider the points $\mathbf{p}_1$ and $\mathbf{y}_3$ as extreme points for which the mid point is the infeasible point $\mathbf{p}_2$. Thus, the new feasible point or right extreme of the line is now the point $\mathbf{p}_2$. Finally, the last point generated is $\mathbf{b}_3$ which can be either lying on or close to the boundary. Condition ((dist_to_boundary(**m**) $\leq \delta$) AND Feasible(**m**)) defines a threshold to stop the process of approaching the boundary. However, the second part of this condition (i.e., "Feasible(**m**)") it is only applied when considering an inequality constraint. In this way, function *BS* guarantees that **m** is in the feasible side regarding the corresponding inequality constraint under consideration. It is worth noticing that parameters **x** and **y** are local to BS, i.e., function BS behaves as a decoder of the pair of feasible and infeasible points passed as parameters. Therefore, the number of "mid_points_between" **x** and **y** before approaching the boundary within a distance less that δ is given by $log_2(r)$ where $r = (dist(\mathbf{x},\mathbf{y})))/\delta$. Thus, the closer to the boundary, the larger $log_2(r)$.

3.2.1 Exploring the boundary region under a general operator

So far, we have shown how a point lying on the boundary **b** can be represented through a pair of points $(\mathbf{x},\mathbf{y})$ with $\mathbf{x} \in \mathscr{F}$ and $\mathbf{y} \in \mathscr{U}$. Now we need to consider the exploration of the search space which, according to our proposal, can be defined as $\mathscr{G} = \{(\mathbf{x},\mathbf{y}) | \mathbf{x} \in \mathscr{F} \subset \mathbb{R}^n \wedge \mathbf{y} \in \mathscr{U} \subset \mathbb{R}^n\}$, that is, the set of pair of points $(\mathbf{x},\mathbf{y})$ as described above. This space can be considered a *genotype space* as known in the area of evolutionary computation. Since each point from $\mathscr{G}$ represents a point on the boundary, it is necessary the application of the decoder represented by function *BS*

Algorithm 1 BS(**x**,**y**: real vector): real vector

```
 1: m: real vector;
 2: repeat
 3:     m = mid_point_between(x, y);
 4:     if Is_on_Boundary(m) then
 5:         return m; { m is a point lying on the boundary }
 6:     end if
 7:     if Feasible(m) then
 8:         x = m;
 9:     else
10:         y = m;
11:     end if
12: until (dist_to_boundary(m)≤ δ) AND (Feasible(m));
13: return m; {The closest point to the boundary according to δ }
```

(see Algorithm 1) to obtain the respective *phenotype*, i.e., the "gene expression" of $(\mathbf{x},\mathbf{y}) \in \mathscr{G}$. Thus, the set $\mathscr{B} = \{\mathbf{b}|\mathbf{b} = BS(\mathbf{x},\mathbf{y})\}$ is conformed by the set solutions on the boundary. Each solution in this set is evaluated by function ϕ, which represents a measure of solutions quality and gives as result an element of set $\mathscr{E} = \{e \in \mathbb{R}|e = \phi(\mathbf{b})\}$. Fig. 4 displays the respective spaces and how they are related with each other by the application of functions *BS* and ϕ, respectively.

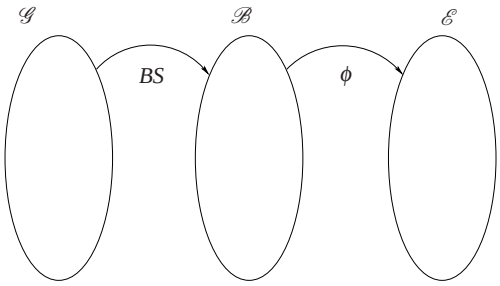

Fig. 4: The search or genotype space ($\mathscr{G}$), phenotype space ($\mathscr{B}$), and space $\mathscr{E}$, and the respective connection through the decoder *BS* and function evaluation ϕ

From the above described, is clear that the search engine must deal with the exploration of space $\mathscr{G}$. Fig. 5 shows a set of three hypothetical points $\{(\mathbf{x}_1,\mathbf{y}_1), (\mathbf{x}_2,\mathbf{y}_2), (\mathbf{x}_3,\mathbf{y}_3)\}$ in $\mathscr{G}$, a problem constraint, and the respective points $\{\mathbf{b}_1,\mathbf{b}_2,\mathbf{b}_3\}$ on the boundary. The application of the general Ω_3 operator on $(\mathbf{b}_1,\mathbf{b}_2,\mathbf{b}_3)$ gives as result a point $\mathbf{b}$ on $\mathscr{F}_{\mathscr{B}}$. To obtain this point on the boundary, an operator χ is applied respectively on the points on $\mathscr{F}$ and $\mathscr{U}$ to obtain a new point on $\mathscr{G}$, i.e., $(\mathbf{x},\mathbf{y})$, from which a new point on the boundary is obtained as displayed in the following:

$$\begin{aligned}\Omega_3(\mathbf{b}_1,\mathbf{b}_2,\mathbf{b}_3) &= \Omega_3(BS(\mathbf{x}_1,\mathbf{y}_1),BS(\mathbf{x}_2,\mathbf{y}_2),BS(\mathbf{x}_3,\mathbf{y}_3))\\ &= BS(\chi_3(\mathbf{x}_1,\mathbf{x}_2,\mathbf{x}_3),\chi_3(\mathbf{y}_1,\mathbf{y}_2,\mathbf{y}_3))\\ &= BS(\mathbf{x},\mathbf{y})\\ &= \mathbf{b}\end{aligned}$$

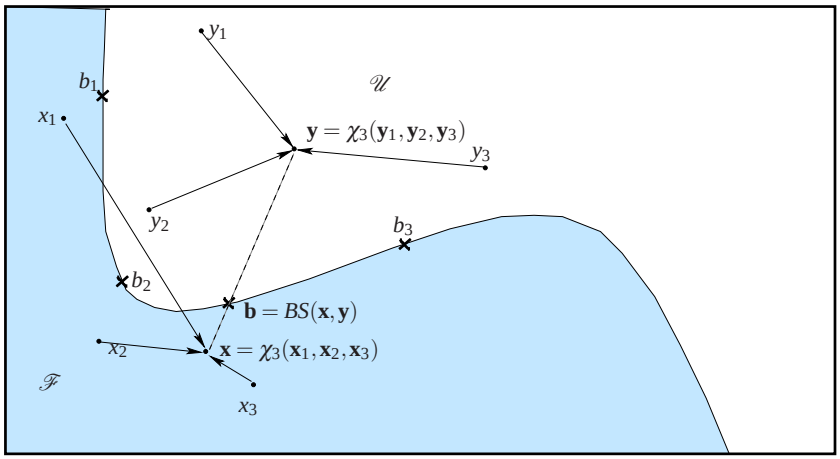

Fig. 5: A set of hypothetical points $\{(\mathbf{x}_1,\mathbf{y}_1),(\mathbf{x}_2,\mathbf{y}_2),(\mathbf{x}_3,\mathbf{y}_3)\}$ in $\mathscr{G}$, a problem constraint, and the set respective points $\{\mathbf{b}_1,\mathbf{b}_2,\mathbf{b}_3\}$ on the boundary. The application of the general 3-ary operator on $(\mathbf{b}_1,\mathbf{b}_2,\mathbf{b}_3)$ gives as result a point $\mathbf{b}$ on $\mathscr{F}_\mathscr{B}$. In fact, the operator Ω_r is a combination of operators (χ) that respectively works on space $\mathscr{F}$ and $\mathscr{U}$, in addition to the decoding function *BS*

Indeed, operator χ could be any exploration operator which will depend on the search engine used to explore space $\mathscr{G}$. For example, from the perspective of evolutionary algorithms, it can be created an initial population of individuals where each one of them represents an element of set $\mathscr{G}$. Therefore, suitable operators to be chosen could be any qualified crossover and/or mutation operators for floating-point representations. A similar approach can be adopted if using another search engine suitable for exploring continuous spaces, e.g., particle swarm optimization, ant colony optimization, differential evolution, immune systems, etc. Sect. 4 describes through three MHs the the boundary approach can be easily implemented with just a few changes when considering different search engines.

3.3 Focusing on the problem constraints

It is important to remember that we are assuming active constraints at the global optimum to proceed with this method which focuses the exploration on the boundary region. However, either using an *ad hoc* or general operator (as the proposed here),

the main difficulty of a boundary operator is concerned with problems with more that one constraint.

Certainly, the simplest case to apply the boundary approach is when the problem has only one constraint which could be either an equality or an inequality constraint. Let us suppose that the problem includes only one constraint, let us say h, then the search engine should proceed by sampling: a) when applying an *ad hoc* operator, a set of solutions laying on the boundary and after that, applying the specific *ad hoc* operators to explore $\mathscr{F}_{\mathscr{B}}$ directly or b) when applying the general operator, a set of pair of points on the *genotype space* $\mathscr{G}$ which each one of them is mapped via function *BS* in to the boundary region, after that, $\mathscr{F}_{\mathscr{B}}$ is indirectly explored through the exploration of space $\mathscr{G}$. In both cases, all solutions generated will be feasible. Two examples of case (a) are certainly given in the Michalewicz et al.'s proposal [17]. For the second case (b), Fig. 5 display a hypothetical problem with one constraint, some points on space $\mathscr{G}$, and the modification of this points which give, via function *BS* a new point on the boundary.

On the other hand, when facing the typical situation in which we have more than one constraint, it is necessary to define an appropriate policy to explore the boundary as efficiently as possible since space $\mathscr{F}_{\mathscr{B}}$ will be now determined by a set of constraints rather than one. In this case, it will be not possible to define any type of boundary operators that make closed $\mathscr{F}_{\mathscr{B}}$ under their application.

In the following we focuses in some alternative to manage this situation considering only the use of the general operator. In fact, the same approach can be applied when considering *ad hoc* operators, however, we believe that this is unlikely due to the difficulty to define them for any type of constraint. Therefore, one possibility is to explore in turn the boundary of each problem constraint. The selection of the constraints to search for can be determined using different methods. If the problem includes at least one equality constraint, such equality constraints are the most appropriate candidates to be selected first. However, a possible search engine could keep focused on a particular constraints over the whole run or may be change from one problem constraint to another depending on a particular condition. In our previous work [13] we defined a simple condition based on a parameter called t_c which counts the number of iterations the algorithm focuses in a particular constraint. However, more complex condition could be considered, for example, taking into account the population diversity or the degree in which some problem constraints are being violated. For the last case, the scheme proposed by Schoenauer and Xanthakis [20] could be adapted and applied when focusing on the boundary region. The proposed scheme consists on a multi-steps evolutionary process based on behavioral memory that considers each problem constraint in turn. The process starts from the first constraint. When the current constraint j is processed, the solutions that violates constraints $j-1,\ldots,1$ are given a zero fitness. Simultaneously, when constraint j is satisfied (according to a threshold ϕ), constraint $j+1$ is then processed. The process continues until all constraints have been considered.

As an illustrative example when facing a problem with more than one constraint, Fig. 6 shows a hypothetical search space determined by three inequality constraints. Let us suppose that the search proceeds starting on constraint g_1. If the visited points

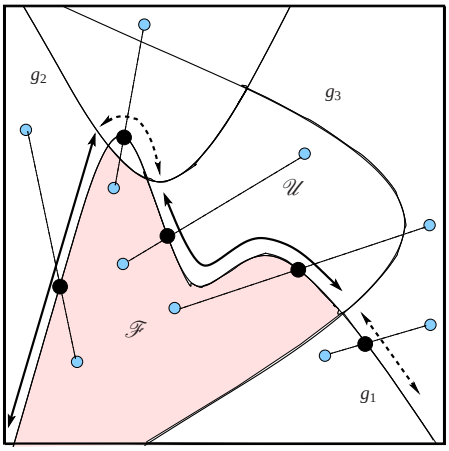

Fig. 6: Feasible search space defined by 3 inequality constraints. The search proceeds on the boundary of constraint g_1, however, some points on the boundary of g_1 are infeasible when considering the whole set of the problem constrains

are on the boundary of $\mathscr{F}$, these points will also satisfy the remaining problem constraints (filled line in Fig. 6). However, the exploration of the boundary with respect to constraint g_1 will eventually produce points violating constraints g_2 and g_3 (dotted line in 6). One of the simplest methods to deal with this situation could be for example, the application of a penalty function for the infeasible solutions. In addition, if g_1 is active at the global optimum, the method will focus the search on the boundary in order to restrict the explored regions of the whole search space. Note however, that other (more sophisticated) constraint-handling techniques can also be adopted. For example, it could be considered the inclusion of the Stochastic Ranking approach [18] to make the comparisons among the solutions generated [14] and thus avoiding the inclusion and tuning of any penalty factor for solutions evaluation. As a manner of showing some concrete examples of the possible application of the boundary approach, in the next section, we focus on its implementation from the perspective of three different search engines: Evolutionary Algorithms, Particle Swarm Optimization, and Ant Colony Optimization.

4 Implementation Issues

This section is aimed to explain in some detail the implementation the boundary approach under the general boundary operator by using three search engines: EAs, PSO, and ACO. Since their implementation under *ad hoc* operators is straightforward, i.e., they do not produce any important change on the respective search engine, we have decided not to include the respective implementation.

The selection of the three mentioned search engines does not follow any kind or priority of one over the remaining ones. In first place, EAs can be considered the more popular MHs used in optimization and particularly in numerical constrained optimization problems. Second, PSO is a more recent MHs which lately have been successfully applied to solve plenty of the state-of-the-art benchmarks for numerical optimization. Finally, we consider ACO as a possible alternative which was chosen for two main reasons: 1) it was the first search engine used to test the boundary approach using a general operator with encouraging results and 2) more advanced version of the ACO metaheuristic have been recently developed which can successfully be applied to problems defined over continuous domains with or without constraints.

Algorithm 2 A general outline of the stochastic ranking algorithm using a bubble-sort like algorithm as defined in [18]. P_f represents the probability of using only the objective function for comparisons when ranking solutions in the infeasible regions of the search space (a value of $0.4 < P_f < 0.5$ was reported as the most appropriate). Parameters N and λ represent respectively the maximum number of sweeps and number of solutions that are ranked by comparing adjacent solutions in at least λ sweeps, and $rnd \in U(0,1)$.

```
1: procedure Sort(var T)
2: I_j = j, ∀j{1,...,λ}
3: for i in 1 : N do
4:     for j in 1 : λ − 1 do
5:         if (T.x_{I_j} == T.x_{I_{j+1}}) || (rnd < P_f)) then
6:             if (T.x_{I_j} > T.x_{I_{j+1}}) then
7:                 swap(I_j, I_{j+1})
8:             end if
9:         else
10:            if (T.x_{I_j} > T.x_{I_{j+1}}) then
11:                swap(I_j, I_{j+1})
12:            end if
13:        end if
14:    end for
15:    if no swap done then
16:        break
17:    end if
18: end for
```

Before giving any detail about the respective algorithms, is worth noticing that all the algorithms were designed including the Stochastic Ranking as complementary handling technique, i.e., the solutions are ranked based on the sorting procedure given in Alg. 2 which receives as argument an structure T containing a set of solutions on $\mathscr{F}_{\mathscr{B}}$ and the respective objective value. Thus, $T.\mathbf{x}_i$ and $T.e_i$ represent respectively the solution i and its objective value. It should be noticed that I_j and I_{j+1} are indexes that point to the structure T. In addition, it is also important to remark that the algorithms described in the following are designed for the general

case, i.e., when the problem includes more than one constraint. However, for the simplest case (problems with only one constraint) the designed algorithms are still applicable by modifying a few lines of code as will be explained for each particular search engine considered. On the other hand, each algorithm includes references to different structures called $T_{\mathscr{F}}$, $T_{\mathscr{U}}$, and $T_{\mathscr{B}}$ which respectively represent a population of solutions in spaces $\mathscr{F}$, $\mathscr{U}$, and $\mathscr{F}_{\mathscr{B}}$ (the same applies to the auxiliary structures called A and A'). Similarly, an structure $T_{\mathscr{E}}$ is used to save the objective value for the respective solutions in $T_{\mathscr{B}}$. Finally, variable 'ctr' indicates the current constraint under consideration, i.e., indicates that the algorithm is currently focusing the exploration on the boundary of constraint 'ctr'. Additional specific structures used by each search engine will be explained when necessary.

Similarly, there exist a set of common functions used through the three algorithms which are described in the following:

- init(): is in charge of obtaining the initial population of points in space $\mathscr{G}$.
- evaluate(): assigns the respective objective value.
- Boundary(): applies function BS() to all the pair of points in $(T_{\mathscr{F}}, T_{\mathscr{U}})$ and returns the respective decoding of those points (the returning structure is usually save in $T_{\mathscr{B}}$).
- change_constraint(): returns a boolean value indicating the decision of focusing the search on a different problem constraint.
- Re_init(): when a change of constraint occurs, this function reinitialize the points in structure $T_{\mathscr{F}}$ and $T_{\mathscr{U}}$ when necessary (e.g., it could be useful a simple perturbation operator here).
- get_next_constraint(): implements the policy for the selection of the next constraint to be considered for exploration. As indicated in the algorithms described further in this section, a possible policy could be the *Round-Robin* policy, however, others (more informed) policies are also possible.
- First$_k$(): returns the first k solutions found in the structure given as parameter.
- Sort(): applies the Alg. 2 to further make the selection of the set first k solutions.
- Update(): selects the best current solutions in $(T_{\mathscr{F}} \oplus A_{\mathscr{F}}, T_{\mathscr{U}} \oplus A_{\mathscr{F}})$[2].

4.1 Boundary by means of EAs

The application of EAs to solve any optimization problems mainly includes a procedure to obtain the initial population, a selection operator, and a set of genetic operators. Alg. 3 describes the more important components of an EA used to solve constrained numerical optimization problems according to the boundary approach. Lines 3 to 5 are aimed to obtain the initial population on space $\mathscr{G}$, the respective points on the boundary and their objective values. It can be observed that the selection process (function Select()) is applied considering $T_{\mathscr{F}}$ and $T_{\mathscr{U}}$, and the respec-

[2] Operator $\oplus$ is defined as follows: $A \oplus B = (a_1,\ldots,a_N) \oplus (b_1,\ldots,b_M) = (a_1,\ldots,a_N,b_{N+1},\ldots,b_{N+M})$ taking into account $T_{\mathscr{B}}$ and the respective objective values.

tive objective value $T_{\mathscr{E}}$. This process produces two intermediate structures $A'_{\mathscr{F}}$ and $A'_{\mathscr{U}}$, i.e., selected points on $\mathscr{G}$ which undergo genetic operators (in two independent steps) as can be observed in lines 14 and 15. The resulting structures are then used as decoders to obtain the respective new points on the boundary (line 17) saved in structure $A_{\mathscr{B}}$. The next step is consists in operate on the ranking (see the SR based function Sort()) of the union of $A_{\mathscr{B}}$ and $T_{\mathscr{B}}$. From this ranking, only the respective best k solutions from space $\mathscr{G}$ will survive for the next generation ('Update()' is used to keep the respective best points on space $\mathscr{G}$).

Algorithm 3 A general outline of EA$_{\mathscr{B}}$

```
 1: t = 0
 2: ctr= initial_constraint // 'ctr' represents the problem constraint under consideration
 3: init(T_F,T_U, ctr);
 4: T_B =Boundary(T_F, T_U)
 5: T_E =evaluate(T_B)
 6: while (stop condition not met) do
 7:     if (change_constraint()) then
 8:         ctr= get_next_ctr(ctr) // The search continues considering another problem constraint,
 9:                                // e.g., following a Round-Robin policy.
10:         A_F =Reinit(T_F, ctr);
11:         A_U =Reinit(T_U, ctr);
12:     else
13:         (A'_F,A'_U) =Select(T_F, T_U, T_E);
14:         A_F =Genetic_ops(A_F, ctr);
15:         A_U =Genetic_ops(A_U, ctr);
16:     end if
17:     A_B =Boundary(A_F,A_U)
18:     T_B =First_k(Sort(T_B ⊕ A_B))
19:     T_E =evaluate(T_B)
20:     Update(T_F,T_U,T_E); { According to the new T_B}
21:     t = t + 1
22: end while
```

When the problem has only one constraint (or only one is considered as in problem *G*02) there is no need to change from one to another constraint, therefore a few code lines can be dismissed, not necessarily dropped, from the general algorithm. In line 2, the 'initial_constraint' is the only problem constraint. The set of lines 7 through 16 are dropped and replaced by lines 13 through 15.

4.2 Boundary by means of PSO

Differently to EA$_{\mathscr{B}}$, a PSO algorithm includes some other additional structures in addition to that used to keep the population or swarm. This particular structures are those representing the respective particles' velocities (called here $V_{\mathscr{F}}$ for space $\mathscr{F}$ and $V_{\mathscr{U}}$ for space $\mathscr{U}$) which let the algorithm explore the respective feasible

and infeasible regions, i.e., the decoder space $\mathscr{G}$. Alg. 4 gives a general outline of a PSO implementing the boundary approach for solving constrained optimization problems. This algorithm follows the principle of PSO design known as "Local Best PSO". For that reason, two proper functions of this PSO version are added, 'Set_the_best_personal_position()' and 'Set_the_best_local_position()'. It can be noticed that they are first applied on $T_{\mathscr{F}}$ and then, on $T_{\mathscr{U}}$. In both cases, the selection on the local and best positions take into account the objective values $T_{\mathscr{E}}$ corresponding to the points they represent on $T_{\mathscr{B}}$.

Algorithm 4 A general outline of lbest PSO$_{\mathscr{B}}$

```
 1: t = 0
 2: ctr= initial_constraint // 'ctr' represents the problem constraint under consideration
 3: init(T_F,T_U, ctr); // Initializes feasible and infeasible swarms
 4: T_B =Boundary(T_F, T_U)
 5: T_E =evaluate(T_B)
 6: while (stop condition not met) do
 7:     for each particle i in T_F do
 8:         Set_the_best_personal_position(T_F(i), T_E(i))
 9:         Set_the_best_local_position(T_F(i), T_E(i))
10:     end for
11:     for each particle i in T_U do
12:         Set_the_best_personal_position(T_U(i), T_E(i))
13:         Set_the_best_local_position(T_U(i), T_E(i))
14:     end for
15:     if (change_constraint()) then
16:         ctr= get_next_ctr(ctr) // The search continues considering another problem constraint,
17:                                // e.g., following a Round-Robin policy.
18:         T_F =Reinit(T_F, ctr);
19:         T_U =Reinit(T_U, ctr);
20:     else
21:         for each particle i in T_F do
22:             Update_velocity(V_F(i), ctr);
23:             A_F(i) = Update_position(T_F(i), V_F(i), ctr);
24:         end for
25:         for each particle i in T_U do
26:             Update_velocity(V_U(i), ctr);
27:             A_U(i) = Update_position(T_U(i), V_U(i), ctr);
28:         end for
29:     end if
30:     A_B =Boundary(A_F,A_U)
31:     T_B =First_k(Sort(T_B ⊕ A_B))
32:     T_E =evaluate(T_B)
33:     Update(T_F,T_U,T_E); { According to the new T_B}
34:     t = t+1
35: end while
```

The exploration stage for PSO$_{\mathscr{B}}$ is accomplished from lines 21 to 28. It can be noticed that the application of two specific PSO steps: 'Update_velocity()' and 'Update_position()'. These two functions are applied on structures $V_{\mathscr{F}}$ and $T_{\mathscr{F}}$ for

the feasible part of space $\mathscr{G}$, and structures $V_{\mathscr{U}}$ and $T_{\mathscr{U}}$ for the infeasible one. In the case of 'Update_position()', it returns the modified point position that can be assigned as in lines 23 and 27. After the obtaining of the new solutions in $A_{\mathscr{F}}$ and $A_{\mathscr{U}}$, the process follows the same steps as for $\text{EA}_{\mathscr{B}}$. Finally, when the problem has only one constraint (or only one is considered as in problem *G*02) there is no need to change from one to another constraint, therefore a few code lines can be dismissed, not necessarily dropped, from the general algorithm. In line 2, the 'initial_constraint' represents the only problem constraint. The set of lines 15 through 29 are replaced by lines 21 through 28.

4.3 Boundary by means of ACO

The last search engine is one based on the ACO metaheuristic. Particularly, we have chosen a recent and advanced version of an ACO algorithm for continuous problems proposed by Socha and Dorigo [22] which is called $\text{ACO}_{\mathbb{R}}$ where the solutions are built by using a probability density distribution (PDF). At step i each ant generates a random number according to a mixture of normal kernels of PDFs $P^i(x_i)$ defined on the interval $a_i \leq x_i \leq b_i$, i.e., a multimodal PDF aimed at considering several subregions of that interval at the same time. These ideas are extensively presented and details concerning implementation issues are given in Socha and Dorigo [22] through algorithm $\text{ACO}_{\mathbb{R}}$ which represents the former ideas proposed by Socha [21] regarding continuous domains. The authors presented an experimental study that considers the application of $\text{ACO}_{\mathbb{R}}$ to a test suite of several unconstrained continuous optimization problems. Alg. 5 displays the main components of the $\text{ACO}_{\mathscr{B}}$, an $\text{ACO}_{\mathbb{R}}$ algorithm that includes the boundary approach for constrained optimization problems. The initialization process includes, in addition, a structure ω which represents a set of weights used as part of the mixture of normal kernels (or PDFs $P^i(x_i)$). Function 'Build_sol()' samples a new set of solutions according the to the respective points in $\mathscr{G}$ and their ranking. Again, structure ω is involved on the sampling process which uses the previous solutions to build an updated model of the PDFs (more details can be found in the description of $\text{ACO}_{\mathbb{R}}$ in Socha and Dorigo [22] as well as in Leguizamón and Coello [13] where the adaptation of $\text{ACO}_{\mathbb{R}}$ for constrained optimization problems is presented). After the sampling of the new solutions in $A_{\mathscr{F}}$ and $A_{\mathscr{U}}$, the process follows the same steps as for $\text{EA}_{\mathscr{B}}$ and $\text{PSO}_{\mathscr{B}}$.

Similarly to the above the search engines, when the problem has only one constraint (or only one is considered as in problem *G*02) there is no need to change from one to another constraint, therefore a few code lines can be dismissed, not necessarily dropped, from the general algorithm. In line 2, the 'initial_constraint' represents the only problem constraint. The set of lines 6 through 15 are replaced by lines 13 through 14.

Algorithm 5 A general outline of $ACO_{\mathscr{B}}$ algorithm

```
 1: t = 0
 2: ctr= initial_constraint // 'ctr' represents the problem constraint under consideration
 3: init(T_F, T_U, ω, ctr);
 4: T_B =Boundary(T_F, T_U)
 5: T_E =evaluate(T_B)
 6: while (stop condition not met) do
 7:     if (change_constraint()) then
 8:         ctr= get_next_ctr(ctr) // The search continues considering another problem constraint,
 9:                                // e.g., following a Round-Robin policy.
10:         A_F =Reinit(T_F, ω, ctr);
11:         A_U =Reinit(T_U, ω, ctr);
12:     else
13:         A_F =BuildSols(T_F, ω, ctr);
14:         A_U =BuildSols(T_U, ω, ctr);
15:     end if
16:     A_B =Boundary(A_F, A_U)
17:     T_B =First_k(Sort(T_B ⊕ A_B))
18:     T_E =evaluate(T_B)
19:     Update(T_F, T_U, T_E); { According to the new T_B}
20:     t = t + 1
21: end while
```

5 Experimental Study

This section presents a (brief) experimental study involving the applications of the boundary approach under the three metaheuristics described in the previous section. The problems considered are conformed by a subset of a well-known testbed non-linear problems used in literarure [16]. Table 1 displays the selected problems and the respectives optimal or best known values (column Opt/Bk).

The study is divided in two parts. The first part (Section 5.2) shows the results of the application of one of the three metaheuritics, namely the ACO approach, by considering two different algorithms. One of them, the original $ACO_{\mathbb{R}}$ enhanced with the stochastic ranking technique to handle constraints ($ACO_{\mathscr{NB}}$). The other one, is the $ACO_{\mathbb{R}}$, but including the boundary approach and stochastich ranking ($ACO_{\mathscr{B}}$). The objective of this study is to show the benefit of the boundary search when included as an alternative constraint handling technique. To do that, we considered just a simple version of the original $ACO_{\mathbb{R}}$ and standard parameter setting for the stochastic ranking in order to better visualize the performance of the algorithm when the search focuses on the boundary between the feasible and infeasible search space. The second part (6) is aimed to compare the performance the boundary approach when implemented under ACO, PSO, and EAs metaheuristics. The respective algorithms are: $ACO_{\mathscr{B}}$, $PSO_{\mathscr{B}}$, and $EA_{\mathscr{B}}$.

Table 1: Set of well-known nonlinear problems

Problem	Opt/BK
G01	-15.00
G02	1.0
G03	0.80619
G04	-30665.539
G05	5126.4981
G06	-6961.8138
G07	24.306209
G09	680.630
G10	7049.2083
G11	0.75
G13	7049.2083
G14	-47.764
G15	961.715
G17	8853.539
G21	193.7783
G23	-400.0025
G24	-5.5079
G25	16.73889

5.1 Parameter Setting

$t_{max} = 10000$, $t_{=}200$, only on active constraints, for all the algorithms

PSO$_{\mathscr{B}}$

$$v_i^j = w \cdot v_i^j + c1 \cdot r1 \cdot (x_i^{lb} - x_i^j) + c2 \cdot r2 \cdot (x_i^{gb} - x_i^j)$$

$$x_i^j = x_i^j + v_i^j$$

$c1 = 0.5$, $c2 = 0.5$, $w = 0.85$, $NP = 50$ (NP is k in Algorithm 3, line 18)

EA$_{\mathscr{B}}$

$p_c = 0.65$, $p_m = 0.01$, $\mu = 50$ (μ is k in Algorithm 4, line 31), $\lambda = 50$, $(\mu + \lambda)$, real vector representation for the individuals, arithmetic crossover, and simple mutation which produces a little change on the value of a partciular dimension vector.

ACO$_{\mathscr{B}}$

$\xi = 0.85$, $q0.1$, $NK = 50$ (NP is k in Algorithm 5, line 17), number of ants set to 50 Referencias a los trabajos previos (ANT 2008) and (en desarrollo)

5.2 Performance Comparison of ACO$_{\mathscr{B}}$ and ACO$_{\mathscr{NB}}$

In this section, we compare the performance of ACO$_{\mathscr{B}}$ and ACO$_{\mathscr{NB}}$. Table 2 shows the results obtained for all the problems studied. Each columns shows respectively for each algorithm, the best found value (BF), average and standard de-

viation (Avg$_{\pm std}$), and number feasible solutions found out of 30 runs. Numbers in boldface (column BF) indicate that the optimal solution was obtained for the respective algorithm. Preliminary results from ACO$_{\mathscr{NB}}$ were obtained by setting $q \in \{0.0001, 0.01, 0.1\}$ and fixing $\xi = 85$. ACO$_{\mathscr{NB}}$ showed a very similar behavior (not results) to ACO$_{\mathscr{B}}$ with respect to parameter q. Even more, the use of a varying q showed to be the more representative for ACO$_{\mathscr{NB}}$ considering the overall performance (quality and number of feasible solutions found).

Table 2: BF in bold means that the optimal value was found for the respective problem. It should be noticed that the results correspond to the setting $q = 0.1$ and $\xi = 0.85$. NA stands for "Not Available".

	ACO$_{\mathscr{B}}$			ACO$_{\mathscr{NB}}$		
Prob.	BF	Avg$_{\pm\sigma}$	#Fea	BF	Avg$_{\pm\sigma}$	#Fea
G01	**-15.00**	-15.00$_{\pm 0}$	30	**-15.00**	-14.10$_{\pm 1.198}$	24
G02	**0.80619**	0.776871$_{\pm 0.025}$	30	0.728079	0.431599$_{\pm 0.1145}$	30
G03	**1.00**	1.00$_{\pm 0}$	30	1.00	1.00$_{\pm 0.0}$	29
G04	**-30665.539**	-30665.539$_{\pm 0}$	30	-30665.58	-30665.57$_{\pm 0.005}$	30
G05	**5126.49**	5135.99$_{\pm 14.54}$	27	5231.96	5231.96$_{\pm 0}$	1
G06	**-6961.814**	-6961.814$_{\pm 0}$	30	**-6961.814**	-6961.814$_{\pm 0}$	2
G07	**24.306**	24.5370$_{\pm 0.240}$	30	24.320	25.041$_{\pm 0.62}$	29
G09	**680.630**	680.630$_{\pm 0}$	30	**680.630**	680.635$_{\pm 0.003}$	30
G10	7049.32	7155.99$_{\pm 94.92}$	30	7049.33	7659.54$_{\pm 596.20}$	29
G11	**0.75**	0.75$_{\pm 0}$	30	**0.75**	0.75$_{\pm 0}$	8
G13	**0.053950**	0.053960$_{\pm 0}$	26	0.097069	0.557918$_{\pm 0.2844}$	11
G14	**-47.760**	-47.686$_{\pm 0.08}$	30	-47.497	-46.315$_{\pm 0.8131}$	10
G15	**961.7151**	961.7157$_{\pm 0}$	30	961.7324	961.8092$_{\pm 0.125}$	3
G17	8863.67	8958$_{\pm 37.64}$	30	9011.91	9018.82$_{\pm 5.582}$	26
G21	193.7860	193.8794$_{\pm 0.09}$	20	NA	NA	NA
G23	-303.54	22.54$_{\pm 145.84}$	17	NA	NA	NA
G24	**5.5080**	5.5080$_{\pm 0}$	30	**5.5080**	5.5080$_{\pm 0}$	28
G25	**16.73889**	16.73889$_{\pm 0}$	30	**16.73889**	16.73889$_{\pm 0}$	7

It can be seen that ACO$_{\mathscr{NB}}$ gives an important number of feasible solutions for some of the problems, however was not able obtain feasible solutions for all the problems (e.g., for $G21$ and $G23$ no feasible solutions were found at all). Taking into account the solved problems by ACO$_{\mathscr{NB}}$ ($G01$, $G06$, $G07$, $G09$, $G11$, $G24$, and $G25$) its performance is inferior to the ACO$_{\mathscr{B}}$ for some of the above problems when considering the number of feasible solutions and/or average values (see problems $G01$, $G06$, $G11$, $G24$, and $G25$). For some of the remaining problems, ACO$_{\mathscr{NB}}$ performs fairly well giving results very close to the optimal ones, however, the main difference with ACO$_{\mathscr{B}}$ is on the average values which shows a less robust algorithm. In addition, for problems $G05$, $G13$, and $G17$; ACO$_{\mathscr{NB}}$ showed the worst performance regarding the solution quality. On the other hand, it can be seen that for problems $G10$, $G17$, and $G21$, ACO$_{\mathscr{B}}$ obtained values very close to the optimal ones, whereas, for problem $G23$ ACO$_{\mathscr{B}}$, showed a poor performance with respect to the solution quality.

It is worth remarking that $ACO_{\mathscr{NB}}$ is a very simple adaptation of the original $ACO_{\mathbb{R}}$ to handling constraints. Despite of that, the results of $ACO_{\mathscr{NB}}$ shows the potential of the ACO approach for continuous problems. Particularly this potential is exploited here by incorporating a boundary approach.

6 Comparison of ACO, PSO, and EAs under the Boundary Approach

This section shows the performance of $ACO_{\mathscr{B}}$, $PSO_{\mathscr{B}}$, and $EA_{\mathscr{B}}$. Table 3 displays the Best Found (BF) value, and the respective average and deviation values ($Avg_{\pm}$) for each one of the algorithms implemented. Numbers in boldface means that the optimal (or best known) value was found. The number of feasible solutions found by the respective algorithms are now showed, however the three algorithms found very similar values to those values displayed in Table 2 for $ACO_{\mathscr{B}}$.

Clearly, there are a subset of problems for which the three algorithms performs identically (see $G01$, $G03$, $G04$, $G06$, $G09$, $G11$, $G24$, and $G25$). For the problem $G05$, the performance is identical with respect to the best value found, however the average values are different but not statistically significant. On the other hand, the results for problems $G02$, $G07$, and $G14$ show that $ACO_{\mathscr{B}}$ outperformed $PSO_{\mathscr{B}}$ and $EA_{\mathscr{B}}$ when considering BF value, however, $EA_{\mathscr{B}}$ is more robust for this problem (see columns BF and $Avg_{\pm\sigma}$). In the case of $G07$ it should be noticed that the three algorithms perform similarly (the average value were not statistically significant).

There exist other particular cases that are analyzed in the following. The results for problem $G10$ show that $ACO_{\mathscr{B}}$ and $EA_{\mathscr{B}}$ outperformed $PSO_{\mathscr{B}}$ considering both, best found and average values. However, $ACO_{\mathscr{B}}$ outperformed $EA_{\mathscr{B}}$ in terms of the average values (statistically significant). For problem $G13$ the above described situation it can also be observed, except that $ACO_{\mathscr{B}}$ found the optima value for this particular problem. A different situation is for problem $G17$ foe which $PSO_{\mathscr{B}}$ found the best solution, however, $ACO_{\mathscr{B}}$ and $EA_{\mathscr{B}}$ showed not statistically significant average value, but better and statistically significant with respect to $PSO_{\mathscr{B}}$. Considering the average value, the three algorithms showed similar performance for problem $G21$, except that $ACO_{\mathscr{B}}$ achieved the best value. Finally, problem $G23$, no one of the three algorithms perform well. $PSO_{\mathscr{B}}$ was not capable of finding any feasible solution, whereas, $ACO_{\mathscr{B}}$ and $PSO_{\mathscr{B}}$ found a few feasible solutions of bad quality.

7 Summary and Future Work

In this chapter we have presented the boundary approach as an alternative approach to explore the boundary between the feasible and infeasible search space for constrained optimization problems. The boundary approach was presented under two perspective, either by using *ad hoc* operators as presented in [17], and a more gen-

Table 3: Performance of $ACO_{\mathscr{B}}$, $PSO_{\mathscr{B}}$, and $EA_{\mathscr{B}}$ on the benchmark problems. BF in boldface means that the optimal value was found for the respective problem. NA stands for "Not Available". The respective parameter setting is described in Section 5.1.

	$ACO_{\mathscr{B}}$		$PSO_{\mathscr{B}}$		$EA_{\mathscr{B}}$	
Prob.	BF	$Avg_{\pm\sigma}$	BF	$Avg_{\pm\sigma}$	BF	$Avg_{\pm\sigma}$
G01	**-15.00**	$-15.00_{\pm 0}$	**-15.00**	$-14.10_{\pm 1.198}$	**-15.00**	$-15.00_{\pm 0}$
G02	**0.80619**	$0.776871_{\pm 0.025}$	0.80512	$0.5939_{\pm 0.06661}$	0.803550	$0.803352_{\pm 0.0001}$
G03	**1.00**	$1.00_{\pm 0}$	**1.00**	$1.00_{\pm 0.0}$	**1.00**	$1.00_{\pm 0.0}$
G04	**-30665.539**	$-30665.539_{\pm 0}$	**-30665.539**	$-30665.539_{\pm 0}$	**-30665.539**	$-30665.529_{\pm 0.01}$
G05	**5126.49**	$5135.99_{\pm 14.54}$	**5126.49**	$5130.74_{\pm 5.44}$	5126.64	$5130.12_{\pm 3.84}$
G06	**-6961.814**	$-6961.814_{\pm 0}$	**-6961.814**	$-6961.814_{\pm 0}$	**-6961.814**	$-6961.814_{\pm 0}$
G07	**24.306**	$24.5370_{\pm 0.240}$	24.375	$25.053_{\pm 0.728}$	24.372	$24.546_{\pm 0.128}$
G09	**680.630**	$680.630_{\pm 0}$	**680.630**	$680.635_{\pm 0.003}$	**680.630**	$680.633_{\pm 0.0}$
G10	7049.3261	$7155.9948_{\pm 94.924}$	7093.0151	$8040.5537_{\pm 972.585}$	7049.333	$7659.540_{\pm 596.20}$
G11	**0.75**	$0.75_{\pm 0}$	**0.75**	$0.75_{\pm 0}$	**0.75**	$0.75_{\pm 0}$
G13	**0.053950**	$0.053960_{\pm 0}$	0.053961	$0.063768_{\pm 0.0103}$	0.053984	$0.069971_{\pm 0.048}$
G14	**-47.760**	$-47.686_{\pm 0.08}$	-47.129	$-43.778_{\pm 1.68}$	47.6724	$47.6708_{\pm 0.0022}$
G15	**961.7151**	$961.7157_{\pm 0}$	**961.7151**	$962.1973_{\pm 1.1407}$	**961.7151**	$961.7149_{\pm 0.0002}$
G17	8863.67	$8958_{\pm 37.64}$	8859.9541	$9019.2519_{\pm 125.776}$	8866.86	$9004.288_{\pm 91.11}$
G21	193.786	$193.879_{\pm 0.09}$	196.392	$201.034_{\pm 3.49}$	193.804	$193.880_{\pm 0.085}$
G23	-303.54	$22.54_{\pm 145.84}$	NA	NA	-177.16	$2.67_{\pm 160.70}$
G24	**5.5080**	$5.5080_{\pm 0}$	**5.5080**	$5.5080_{\pm 0}$	**5.5080**	$5.5080_{\pm 0}$
G25	**16.73889**	$16.73889_{\pm 0}$	**16.73889**	$16.73889_{\pm 0}$	**5.5080**	$5.5080_{\pm 0}$

eral operator as proposed in previous works [13, 14] as possible alternatives to be applied when facing constrained problems. In addition, three different search machines (EAs, PSO, and ACO) were considered to show how the boundary approach could be implemented without producing major modifications on the original algorithms. Furthermore, it is worth noticing that the boundary approach is an interesting mechanism that could applied to many constrained optimization problems, particularly this observation is true for the 'general boundary operator' described in detail here and proposed in earlier works.

As presented in this chapter, the boundary approach can be used alone (as a constraint-handling techniques itself) or in combination with a complementary constraint-handling technique like penalty functions or any other like Stochastic Ranking which is used here to display three algorithms due to his simplicity and for being an efficient and very well known technique. The three algorithms (i.e., $EA_{\mathscr{B}}$, $PSO_{\mathscr{B}}$, and $ACO_{\mathscr{B}}$) presented here as a guidelines show that the boundary approach is flexible enough to be implemented under different search machines.

Finally, is important to remark that the boundary approach (seen as having the possibility of using boundary operators) can be considered when implementing a more general constraint-handling technique under some search engine. Particularly when facing problems with an several constraints. For example, the general operator (as defined here) needs two points, one from the feasible region and the other one form the infeasible one. Let us suppose that a MHs implement a constraint-handling technique to explore the whole space $\mathscr{F}$. However, the boundary of a particular

constraint could be approached by considering points from both, the feasible and infeasible one with respect to that constraint. By this way, the generation of solutions laying on the boundary could help to quickly reach (in a controlled manner) the boundary region. Nevertheless, there are still place to consider alternative ways of implementing a *general boundary operator* different form the proposed in this work.

Acknowledgements The first author acknowledges support from Universidad Nacional de San Luis and the ANPCYT (National Agency for Promotion of Science and Technology). The second author acknowledges support from CONACyT project no. 45683-Y.

References

1. Th. Bäck, D. B. Fogel, and Z. Michalewicz, editors. *Handbook of Evolutionary Computation*. Oxford University Press, New York, and Institute of Physics Publishing, Bristol, 1997.
2. C. Coello Coello. List of references on constraint-handling techniques used with evolutionary algorithms. http://www.cs.cinvestav.mx/-constraint/.
3. C. Coello Coello. Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: A surv ey of the state of the art. *Computer Methods in Applied Mechanics and Engineering*, 191(11-12):1245–1287, January 2002.
4. D. Corne, M. Dorigo, and F. Glover, editors. *New Ideas in Optimization*. McGraw-Hill International, 1999.
5. M. Dorigo and T. Stützle. *Ant Colony Optimization.* Mit-Press, 2004.
6. A.P. Engelbrecht. *Fundamentals of Computational Swarm Intelligence*. Jon Wiley & Sons, Ltd, 2005.
7. M. Schütz G. Leguizamón, Z. Michalewicz. An ant system for the maximum independent set problem. In *Proceedings of the 2001 Argentinian Congress on Computer Science*, pages 1027–1040, El Calafate, Argentina,, 2001.
8. F. Glover and G.A.Kochenberg, editors. *Handbook of Metaheuristics*. Kuwler Academic Publishers, London, 2003.
9. J.Gottlieb. Evolutionary algorithms for multidimensional knapsack problems: the relevance of the boundary of the feasible region. In Wolfgang Banzhaf, Jason Daida, Agoston E. Eiben, Max H. Garzon, Vasant Honavar, Mark Jakiela, and Robert E. Smith, editors, *Proc. of the Genetic and Evolutionary Computation Conf. GECCO-99*, page 787, San Francisco, CA, 1999. Morgan Kaufmann.
10. A. J. Keane. Experiences with optimizers in structural design. In I. C. Parmee, editor, *Proceedings of the Conference on Adaptive Computing in Engineering Design and Control*, pages 14–27, Plymouth, UK, 1994. University of Plymouth.
11. J. Kennedy and R Eberhart. Particle swarm optimization. In *IEEE International Conference on Neural Networks*, volume IV, pages 1942–1948, Perth, Australia, 1995. IEEE Service Center.
12. J. Kennedy, R.C. Eberhart, and Y. Shi. *Swarm Intelligence.* Morgan Kaufmann, 2001.
13. G. Leguizamón and C. Coello Coello. Boundary search for constrained numerical optimization problems in aco algorithms. In Marco Dorigo, Luca Maria Gambardella, Mauro Birattari, Alcherio Martinoli, Riccardo Poli, and Thomas Stützle, editors, *ANTS Workshop*, volume 4150 of *Lecture Notes in Computer Science*, pages 108–119. Springer, 2006.
14. G. Leguizamón and C. Coello-Coello. A Boundary Search based ACO Algorithm Coupled with Stochastic Ranking. In *2007 IEEE Congress on Evolutionary Computation (CEC'2007)*, pages 165–172, Singapore, September 2007. IEEE Press.
15. G. Leguizamón and Z. Michalewicz. A New Version of Ant System for Subset Problems. In *Proceedings of the 1999 Congress on Evolutionary Computation*, pages 1459–1464. IEEE Press, Piscataway, NJ, 1999.

16. J. J. Liang, T. P. Runarsson, E. Mezura-Montes, M. Clerc, P. N. Suganthan, C. Coello Coello, and K. Deb. Problem Definitions and Evaluation Criteria for the CEC. Technical report, Special Session on Constrained Real-Parameter Optimization, School of Electrical and Electronic Engineering Nanyang Technological University, available at http://www.ntu.edu.sg/home5/lian0012/cec2006/technical_report.pdf, Singapore, 2006.
17. Z. Michalewicz, G. Nazhiyath, and M. Michalewicz. A note on usefulness of geometrical crossover for numerical optimization problems. In Lawrence J. Fogel, Peter J. Angeline, and Thomas Bäck, editors, *Evolutionary Programming V: Proc. of the Fifth Annual Conf. on Evolutionary Programming*, pages 305–311, Cambridge, MA, 1996. MIT Press.
18. T.P. Runarsson and X. Yao. Stochastic ranking for constrained evolutionary optimization. *IEEE Transactions on Evolutionary Computation*, 4(3):284–294, 2000.
19. M. Schoenauer and Z. Michalewicz. Evolutionary computation at the edge of feasibility. In Hans-Michael Voigt, Werner Ebeling, Ingo Rechenberg, and Hans-Paul Schwefel, editors, *Parallel Problem Solving from Nature – PPSN IV*, pages 245–254, Berlin, 1996. Springer.
20. M. Schoenauer and S. Xanthakis. Constrained GA optimization. In Stephanie Forrest, editor, *Proc. of the Fifth Int. Conf. on Genetic Algorithms*, pages 573–580, San Mateo, CA, 1993. Morgan Kaufmann.
21. K. Socha. ACO for continuos and mixed-variable optimization. In M. Dorigo, M. Birattari, C. Blum, L. M. Gambardella, F. Mondada, and T. Stützle, editors, *Ant Colony Optimization and Swarm Intelligence, 4th International Workshop, ANTS 2004*, volume 3172 of *LNCS*, pages 25–36. Springer-Verlag, Berlin, Germany, 2004.
22. K. Socha and M. Dorigo. Ant colony optimization for continuous domains. *European Journal of Operational Research*, 185(3):1115–1173, to be published in March 2008. http://dx.doi.org/10.1016/j.ejor.2006.06.046.
23. Z.Y. Wu and A.R. Simpson. A self-adaptive boundary search genetic algorithm and its application to water distribution systems. *Journal of Hidraulic Research*, 40(2):191–203, 2002.