Boundary Search for Constrained Numerical Optimization Problems in ACO Algorithms

Guillermo Leguizamón[1] and Carlos A. Coello Coello[2]

[1] LIDIC - Universidad Nacional de San Luis, Ejército de los Andes 950
(5700) San Luis, ARGENTINA
`legui@unsl.edu.ar`
[2] Evolutionary Computation Group (EVOCINV) at CINVESTAV-IPN
Electrical Engineering Department, Computer Science Section
Av. IPN No. 2508 Col. San Pedro Zacatenco México D.F. 07300, MÉXICO
`ccoello@cs.cinvestav.mx`

Abstract. This paper presents a novel boundary approach which is included as a constraint-handling technique in an ant colony algorithm. 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. Our proposed technique precisely focuses the search on the boundary region and can be either used alone or in combination with other constraint-handling techniques depending on the type and number of problem constraints. For validation purposes, an ant algorithm is adopted as our search engine. We compare our proposed approach with respect to constraint-handling techniques that are representative of the state-of-the-art in constrained evolutionary optimization using a set of standard test functions.

1 Introduction

One of the first ACO extensions to operate on continuous spaces can be found in Bilchev et al. [1] in which the whole search space is discretized in order to represent a finite number of search directions. This approach was validated using a small set of constrained problems. Since then, several other researchers have proposed schemes to apply the ACO algorithm to continuous search spaces (see for example [2–4]). However, all of these approaches only deal with unconstrained optimization problems.

In this paper we introduce a boundary approach for solving nonlinear constrained problems, which is presented as a possible extension of ACO algorithms in continuous search spaces. It is worth noting, however, that our proposal can be coupled to other metaheuristics (e.g., particle swarm optimization or an evolutionary algorithm), and it is expected to be highly competitive in problems with active constraints. Our ACO approach is mainly based on the work of Bilchev et al. [1]. The reason for not adopting one of the more recent ACO extensions for continuous search spaces is that, as indicated before, none of them deals with constrained optimization problems.

2 Constraint-Handling and Boundary Search

Michalewicz et al. [5] is one of the first papers on boundary search through the use of evolutionary algorithms. The efficiency of this approach was shown by using two constrained optimization problems: Keane's function (also known as $G02$) [6] and another function with one equality constraint (also known as $G03$). For solving $G02$ the authors proposed two genetic operators: the *geometrical* crossover and a special mutation operator. Both operators generate offspring lying on the boundary between the feasible and infeasible search space. Similarly for $G03$, they proposed the *spherical* crossover which only generates points on the surface of the sphere given as the only constraint. Schoenauer et al. [7] 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 depends on the surface representation: curves-based, plane-based, and parametric representation. Wu et al. [8] 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. 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 so far by other researchers only include problems with one or two constraints (e.g., $G02$ and $G03$[3], respectively). In these cases, 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 constraints, 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. This was precisely the motivation for the research reported in this paper.

3 An Alternative Boundary Search Approach

We are interested in solving 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 $\mathbf{x} \in \mathcal{F} \subset \mathcal{S}$. The set $\mathcal{S} \in R$ defines the search space and sets $\mathcal{F} \subseteq \mathcal{S}$ and $\mathcal{U} = \mathcal{S} - \mathcal{F}$ define the *feasible* and *infeasible* search spaces, respectively. The search space $\mathcal{S}$ is defined as an *n*-dimensional rectangle in $\mathbb{R}^n$ (domains of variables defined by their lower and upper bounds): $l(i) \leq x_i \leq u(i)$ for $1 \leq i \leq n$ whereas the feasible set $\mathcal{F}$ is defined by the intersection of $\mathcal{S}$ and a set of additional $m \geq 0$ constraints: $g_i \leq 0$, for $j = 1, \ldots, q$ and $h_j = 0$ for $j = q+1, \ldots, m$. At any point $\mathbf{x} \in \mathcal{F}$, 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 $\mathcal{F}$.

3.1 The boundary operators

We propose here a general boundary operator which is based on the notion that each point $\mathbf{b}$ of the boundary region can be represented by means of two different points

---

[3] Keane's function can be considered as having one constraint since one of them is ignored focusing the search only on the active constraint.

$\mathbf{x}$ 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 $\mathbf{y}$. Figure 1 shows a hypothetical search space including the feasible and infeasible (shadowed area) 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 $(\mathbf{x}_4, \mathbf{y}_4)$.

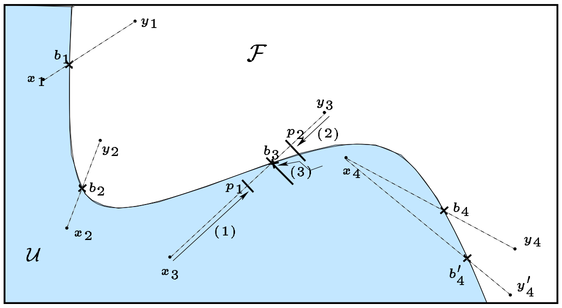

Fig. 1. Given one feasible and one infeasible point, the respective point lying on the boundary can be easily reached by using a simple binary search. .

The binary search applied to each pair of points $(\mathbf{x}, \mathbf{y})$ is achieved following the steps described in function BS (see Figure 2). For example, a possible application of this process can be seen in Figure 1 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 infeasible. Consequently, the left side of the straight line $(\mathbf{x}_3)$ is moved to point $\mathbf{p}_1$. In the next step ((2)) we consider the points $\mathbf{p}_1$ and $\mathbf{y}_3$ as extreme points for which the mid point is the feasible 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$(\mathbf{m}) > \xi$) defines a threshold to stop the process of approaching the boundary. It is worth noticing that parameters $\mathbf{x}$ and $\mathbf{y}$ 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" $\mathbf{x}$ and $\mathbf{y}$ before approaching the boundary within a distance less that ξ is given by $log_2(r)$ where $r = (dist(\mathbf{x}, \mathbf{y})))/\xi$. Thus, the closer to the boundary, the larger $log_2(r)$.

So far, we have shown how a point lying on the boundary can be represented through a pair of points. Now we need to consider the exploration of the search space. For example, from the perspective of evolutionary algorithms, the candidate operators are crossover and mutation. However, for the ACO approach we only adopt a mutation-like operator. Here, a pair of points is considered (one feasible and one infeasible). Alternatively, any of these two points could be modified. For example, we can consider the pair of points $(\mathbf{x}_4, \mathbf{y_4})$ in Figure 1 which represents point $\mathbf{b}_4$ on the boundary. In this case, the feasible point $\mathbf{y}_4$ can be modified giving as a result a point $\mathbf{y}'_4$ in the feasible

```
function BS($\mathbf{x}$,$\mathbf{y}$: real vector): int
begin
  do
    $\mathbf{m}$ = mid_point_between($\mathbf{x}, \mathbf{y}$);
    if ( Is_on_Boundary($\mathbf{m}$) ) return $\mathbf{m}$; /* $\mathbf{m}$ is a point lying on the boundary */
    if ( Feasible($\mathbf{m}$) ) $\mathbf{x} = \mathbf{m}$; else $\mathbf{y} = \mathbf{m}$;
  while ( dist_to_boundary($\mathbf{m}$)$>$ $\xi$ );
  return mid_point_between($\mathbf{x}, \mathbf{y}$); /* The closest point to the boundary */
end
```

Fig. 2. Given one feasible and one infeasible point, function BS returns either a point on the boundary or one which is close enough to the boundary according to a parameter ξ.

search space. After this process, the new point lying on the boundary is obtained by decoding $(\mathbf{x}_4, \mathbf{y}'_4)$, which gives us $\mathbf{b}'_4$.

3.2 The proposed method

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. For the last case, it is important to remember that we are assuming active constraints at the global optimum to proceed with this method where the search is always performed on the boundary of the space defined by any of the constraints.

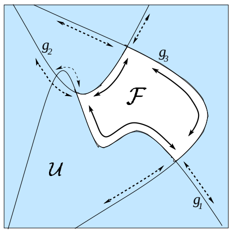

Fig. 3. Feasible search space defined by 3 inequality constraints.

For 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. One possibility is to explore in turn the boundary of each 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. In order to show the robustness of our method in the absence of information about the active constraints of a problem, we will show in our experimental study (see Section 5) a more general approach to apply the boundary operators. As an illustrative example, Figure 3 shows a hypothetical search space determined by three inequality constraints. Let's suppose that the search proceeds starting on constraint g_1 by using the boundary operator O_{g_1} (filled line in Figure 3). The application of this operator will eventually produce points violating constraints g_2 and g_3 (dotted line in Figure 3). One of the simplest methods to deal with this situation is 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.

4 Boundary Approach in ACO algorithms

A possible design to apply the ACO approach in continuous search problems is by discretizing the continuous search space in some way. In this work we use a discrete structure to represent a set of different points spread on the search space. These points are called *directions*, following Bilchev et al.'s proposal. The discrete structure can be seen as a set $\{d_1, d_2, ..., d_k\}$, where k is a parameter for the number of directions, i.e., the population size. Each direction d_i is represented as a real n-dimensional vector. A general outline of the ACO algorithm is shown in Figure 4. It is worth remarking that the original proposal [1] for ACO in continuous domains is used to proceed with the local exploration after a genetic algorithm has finished with the global search. However, the algorithm proposed here, is in charge of performing the entire search process. More precisely, our ACO algorithm starts with a set of k pairs of points $(\mathbf{x}, \mathbf{y})$ randomly generated with $\mathbf{x} \in \mathcal{F} \iff \mathbf{y} \in \mathcal{U}$ (when considering an equality constraint, $\mathbf{z} \in \mathcal{F}$ iff $h(\mathbf{z}) \leq 0$; otherwise, $\mathbf{z} \in \mathcal{U}$) . In addition, a value $0 \leq R \leq 1$ is considered to define the extent of the search interval with respect to each variable. Parameter R starting at value 1 will vary down to 0 on each iteration as described later in this section.

```
ACO algorithm
begin
    t = 0; initialize $A(t)$; evaluate $A(t)$;
    while ( stop condition not met ) do
    begin
        t = t + 1
        update_trail; reallocate_ants $A(t)$;
        evaluate $A(t)$;
    end
end
```

Fig. 4. General outline of the ACO algorithm for continuous problems adopted in this paper.

The ACO algorithm displayed in Figure 4 works as follows: `initialize A(t)` "distributes" N_a ants on the k directions, where $N_a > k$ in order to allocate one or more ants to the same direction. Each ant randomly generates one possible solution, i.e. a pair $(\mathbf{x}, \mathbf{y})$ with $x \in \mathcal{F}$ and $y \in \mathcal{U}$; `evaluate A(t)` obtains the objective value for the new points generated; *update_trail* is in charge of accumulating pheromone trial in each direction proportionally to the quality of the objective function values found in the corresponding direction, i.e., $\tau_d = (1 - \rho) \cdot \tau_d + \Delta\tau_d$ where $\Delta\tau_d$ is a value proportional to the best objective value on direction d and $0 \leq \rho \leq 1$ is the pheromone trail evaporation rate; *reallocate_ants A(t)* redistributes the population of ants on the k directions, proportionally to the accumulated pheromone trail values. Thus, the ants on direction $d \in \{1, \dots, k\}$ are in charge of searching in the neighborhood of the respective boundary feasible point on direction d. The changes on the values of ratio R to control the extent of the search interval for each dimension can be implemented as $\Delta_R(t) = R(1 - r^{(1-t/T)})$ where r is a random number in the range $[0..1]$; T is the maximum number of iterations. Consequently, the value $\Delta_R(t)$ falls in the range $[0..R]$ and gets closer to 0 as the elapsed number of iterations t increases.

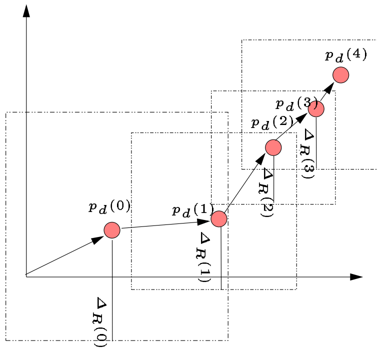

Fig. 5. Sequence of points generated in the search space and limited to the extent of values on each dimension.

Figure 5 represents the successive points (a 2-dimensional vector) on direction j starting from point $p_d(t)$ for $t \in \{0, 1, 2, 3, 4\}$. Thus, a square represents the neighborhood for a particular point. Following the algorithm, at the first iteration a fixed number of ants are distributed on the k directions, i.e., the ants that were allocated to direction d at the iteration $t = 0$ will start the search from point $p_d(0)$. For example, in Figure 5, $p_d(0)$ is the starting point on direction d, $p_d(1)$ is the best point found by the ants allocated to direction d by using $\Delta_R(0)$[4]. As t increases, new regions of the search space are independently explored on each direction. For our example, the remaining successively generated points on direction d are $p_d(2), p_d(3), p_d(4)$, and so on. Thus, the ACO algorithm can be seen as a trajectory approach which simultaneously searches on different directions and exploits the past experience to guide the search towards the most promising regions according to the quality of the results. Furthermore, the accumulated pheromone trail will decrease on the direction that produces low-quality solutions due to the effects of the evaporation process focusing the ants' attention on more promising regions of the feasible search space. In order to avoid premature convergence of the

---

[4] It should be noticed that $\Delta_R(t)$ is not a monotonically decreasing value.

algorithm, a potentially useful direction can remain as an alternative search region by bounding with lower and upper values the amount of pheromone trial in each direction following the principle of the $\mathcal{MM}$AS algorithm [9]. The main characteristics of this method include two abstraction levels:

1. *individual search*: involves the strategy followed by each ant to search in its neighborhood (in our case, a mutation-like operator).
2. *cooperation*: involves information exchange among the ants in order to guide the search to certain regions of the search space. This information is represented by the pheromone trial structure (τ) where τ_j represents the accumulation of pheromone trail on direction j. The distribution of the ants on the different directions is achieved by the formula: $P_d(t) = \frac{\tau_d(t)}{\sum_{h=1}^{k} \tau_h(t)}$

5 Analysis of Results

The application of our approach (called ACO$_{\mathcal{B}}$) requires minimum changes when applied to the different test cases considered: the objective function, the number of variables, the range of each variable, and the constraints. However, the policy to determine on which constraint the search should focus needs to be considered when the problems have more than one constraint: a) we can focus the search on all the constraints, but considering one constraint in turn by controlling the change through a particular condition (S$_{all}$), b) similar to the previous alternative but considering only the active constraints (S$_{act}$), or c) just considering one constraint during the whole run (S$_c$ where $c \in \{1, \ldots, m\}$). These three ways of exploring the search space are presented first in our experimental study in order to analyze the performance of the ACO$_{\mathcal{B}}$ on each of the considered problems. In our experiments, the condition to produce a change on the search from one to another constraint is given by an elapsed number of iterations and it is represented by the parameter t_c. In addition, for problems with more than one constraint, we incorporate a penalty function of the form:

$$\phi(x, \mu) = f(x) + \mu(t)\left(\sum_{j=1}^{q} \max\{0, g_j(x)\} + \sum_{j=q+1}^{m} |h_j(x)|\right) \qquad (1)$$

where $\mu(t)$ is a dynamic penalty factor which could change as t, the elapsed iteration, increases with $\mu(0) \leq \mu(1) \leq \mu(2) \ldots \leq \mu(T)$. Alternatively, the penalty factor can be fixed throughout the run, i.e., $\mu(t) = \mu_0$ for all $1 \leq t \leq T$. Regardless of the penalty function adopted, it is worth remarking that each solution is always lying on the boundary of the feasible space corresponding to the constraint under consideration. Note that this approach was adopted due to its simplicity, since our interest was to assess the advantages of our proposed approach. However, other constraint-handling techniques are evidently possible. The parameter values used in the experimental study are the following: 50 ants (population size), 20 directions (number of points), maximum number of iterations 30000, the evaporation rate $\rho = 0.5$, $t_c > 0$ is the number of iterations the ACO$_{\mathcal{B}}$ focuses on one constraint in turn. When $t_c = 0$, the ACO$_{\mathcal{B}}$ focuses on only one constraint throughout the whole run ($t_c = 200$). The penalty factor

Table 1. Results for problems $G02$ (Keane's function) and $G03$.

No. of Variables (n)	BF	Mean	Std	Worst	# Fea	Mean(#E)
	Problem G02					
20	0.8036190867	0.8025656939	0.0032	0.7930839658	30	29500
50	0.8352618814	0.8339309692	0.0021	0.8259508014	30	35900
100	0.8456841707	0.8446936011	0.0007	0.8423509002	30	46700
	Problem G03					
20	1.0	1.0	0.0	1.0	30	140000
50	1.0	1.0	0.0	1.0	30	389500

$\mu(t)$ was experimentally determined for each particular problem and is shown in the corresponding tables of results. The ACO$_{\mathcal{B}}$ was executed 30 times with different seeds for each parameter combination. The problems studied include a set of well-known test cases traditionally adopted in the specialized literature: $G01$, to $G13$ [10].

5.1 Study of the application of ACO$_{\mathcal{B}}$

We have divided the presentation of the results into two groups according to the following criteria: the first group, is displayed in Tables 1 and 2. Table 1 includes two special cases since they where the first problems on which the boundary approach was applied (problems $G02$ and $G03$). In addition, these problems have one and two constraints respectively. However, the second constraint of problem $G02$ is not considered since it is not active at the best known value. The columns in this table show the setting for the number of variables, the best value found (BF), Mean, Standard Deviation (Std), Worst, number of feasible solutions out of 30 runs (#Fea), and the mean number of evaluations to get the best value found (Mean(#E)). On the other hand, Table 2 shows two problems both of which include one equality constraint (problems $G11$ and $G25$). Accordingly, no penalty values (μ) need to be applied for this first group of problems. In the remaining tables, the column "No. of variables" is replaced by "Cnst", indicating the criteria adopted to proceed with the boundary search, i.e., S_c ($c \in \{1, \ldots, m\}$), S_{act}, or S_{all}. In addition, the best known or global optimum value for each problem is shown in parenthesis.

We tested $G02$ setting the number of variables as $n = 20, 50$, and 100. ACO$_{\mathcal{B}}$ succeeded in finding the best known value for $n = 20$ [11]. In addition, it was able to find a better quality result than the best objective reported in [7] where $n = 50$ and $f(\mathbf{x}^*) = 0.831937$. For $n = 100$, we found 0.8456841707 as the best value in our experimental study. Also, it is worth remarking that all the solutions found were feasible for all n and very similar among themselves as can be observed in the columns Mean, Std, and Worst. With respect to problem $G03$, we considered $n = 20$ and $n = 50$ variables. ACO$_{\mathcal{B}}$ found the optimum feasible solution for both cases in all runs. Similarly to $G03$, the remaining problems of this group ($G11$ and $G25$), our approach reached the optimum in all cases.

The second group of the test cases is conformed by some problems having more than one constraint which have been frequently used in the specialized literature: $G01$,

Table 2. For problems G11 and G25 it is unnecessary to use a penalty factor.

Cnst.	BF	Mean	Std	Worst	# Fea.	Mean(#E)
	Problem G11 (0.75)					
S_1	0.75	0.75	0.0	0.75	30	70400
	Problem G25 (16.73889)					
S_1	-16.73889	-16.73889	0.0	-16.73889	30	10600

$G04$, $G05$, $G06$, $G07$, $G09$, $G10$, and $G13$. Also we include problem $G24$ [12] in this subgroup. Only for $G10$, we adopted a dynamic penalty ($\mu(t) = 1.05 \times \mu(t-1)$ for $t = 0, 1, \cdots t_{max}$, with $\mu(0) = 200000$). The static penalty factors adopted for the remaining problems are (i.e., for $t = 0, 1, \cdots T$): $G01$, $\mu(t) = 1000$; $G04$, $\mu(t) = 800000$; $G05$, $\mu(t) = 10$; $G06$, $\mu(t) = 10000$; $G07$, $\mu(t) = 20000$; $G09$, $\mu(t) = 2000$, $G13$, $\mu(t) = 0.2$; and $G24$, $\mu(t) = 1000$. The results for this group of problems are displayed in Tables 3 and 4. It must be noticed that these problems include different numbers and complexities of the equality and inequality constraints which are active at the best known or optimum solution. As indicated in column "Cnst.", each row shows the results when ACO$_{\mathcal{B}}$ is applied one of the following criteria: search exclusively on constraint j (S_j, $j = 1, \ldots, m$), on all the active constraints in turn (S_{act}), and over all the constraints in turn (S_{all}). For example, problem $G01$ has 6 active constraints. Accordingly, ACO$_{\mathcal{B}}$ performs optimally when searching on those active constraints. Similarly, the algorithm succeeded in finding the optimal solution when using both S_{act} and S_{all}. However, its performance slightly decays when searching on the non active constraints as could be expected. This situation is more dramatic for problem $G04$ which has two active constraints. In this case, ACO$_{\mathcal{B}}$ only finds the optimum solution when searching on the respective active constraints and S_{act}. Although strategy S_{all} fails in finding any feasible solution, this strategy worked well for all the other problems considered. A similar situation can be seen for problem $G05$ which has three equality constraints. Accordingly, ACO$_{\mathcal{B}}$ finds a high quality solution for this problem (very near to the optimal one) when searching on the equality constraints, S_{act}, and S_{all}. On the other hand, problem $G06$ has two inequality constraints which are active at the optimum. ACO$_{\mathcal{B}}$ performs optimally for this problem by following any of the three applicable strategies: S_1, S_2, and S_{act}. The last problem in Table 3 has six active constraints and ACO$_{\mathcal{B}}$ performs similarly to $G01$ since the best results were obtained when searching on the active constraints or by using S_{act} or S_{all}.

Problem $G09$ has two active constraints for which ACO$_{\mathcal{B}}$ found the optimum value. However, searching on the non active constraints can give results far from the expected value (see S_2 and S_3). $G10$ constitutes one of the most difficult test cases not only for our approach, but also for any other constraint-handling technique. ACO$_{\mathcal{B}}$ found feasible solutions with all the search strategies except for S_5 and S_6. Note the small number of feasible solutions found for this problem, as well as the large standard deviation value produced (with respect to the deviations of the other problems). Another interesting problem is $G13$ whose feasible search space is defined by three nonlinear equality constraints. For this problem ACO$_{\mathcal{B}}$ found the optimal solution following either of the four applicable search strategies. Finally, it can be seen that ACO$_{\mathcal{B}}$ performs optimally

Table 3. Results for problems $G01$, $G04$, $G05$, $G06$, and $G07$.

Cnst.	BF	Mean	Std	Worst	# Fea.	Mean(#E)	Cnst.	BF	Mean	Std	Worst	# Fea.	Mean(#E)
	Problem G01 (-15.00)							Problem G07 (24.306)					
S_1	-15.00	-14.99	0.001	-14.996	30	274800	S_1	24.37	29.59	4.83	42.97	30	70000
S_2	-15.00	-14.96	0.012	-14.995	30	159720	S_2	24.51	35.10	23.02	121.56	30	133600
S_3	-15.00	-14.99	0.001	-14.965	30	381800	S_3	24.56	28.31	5.54	50.83	30	83600
S_4	-14.27	-13.54	.38	-13.18	29	544400	S_4	24.79	54.17	70.46	380.03	30	83600
S_5	-13.84	-13.48	0.32	-13.04	25	433800	S_5	24.52	34.52	16.39	77.19	30	85000
S_6	-14.22	-13.39	0.47	-13.00	26	407200	S_6	24.79	31.12	6.46	48.40	30	720800
S_7	-15.00	-14.78	0.2	-14.65	26	213400	S_7	33.08	38.86	4.01	46.53	30	71200
S_8	-15.00	-14.74	0.49	-14.46	27	723400	S_8	41.03	46.86	20.92	127.06	30	260200
S_9	-15.00	-14.67	0.76	-13.08	30	454800							
S_{act}	-15.00	-15.00	0	-15.00	30	81400	S_{act}	24.37	24.64	0.15	24.92	30	35600
S_{all}	-15.00	-15.00	0	-15.00	30	104000	S_{all}	24.38	24.76	0.16	25.22	30	56000
	Problem G04 (-30655.539)							Problem G05 (5126.49)					
S_1	-30665.539	-30665.357	0.04	-30665.157	30	512200	S_1	-	-	-	-	-	-
S_2	-	-	-	-	-	-	S_2	-	-	-	-	-	-
S_3	-	-	-	-	-	-	S_3	5126.50	5133.29	9.284	5147.81	6	100800
S_4	-	-	-	-	-	-	S_4	5126.51	5134.70	11.219	5164.91	11	340000
S_5	-	-	-	-	-	-	S_5	5126.68	5130.55	3.656	5136.08	11	180000
S_6	-30655.539	-30665.302	0.01	-30665.290	30	326400							
S_{act}	-30655.539	-30655.539	0.001	-30655.539	30	19800	S_{act}	5126.50	5138.37	8.20	5132.14	6	94000
S_{all}	-	-	-	-	-	-	S_{all}	5126.50	5143.77	10.60	5163.56	5	135800
	Problem G06 (6961.81)												
S_1	-6961.79	-6961.71	0.075	-6169.54	11	122600							
S_2	-6961.81	-6961.72	0.097	-6961.34	25	103000							
S_{act}	-6961.81	-6961.74	0.070	-6961.71	25	80000							

Table 4. Results for problems $G09$, $G10$, $G13$, and $G24$.

Cnst.	BF	Mean	Std	Worst	# Fea.	Mean(#E)
	Problem G09 (680.63)					
S_1	680.63	680.66	0.10	681,29	30	80400
S_2	1664.00	1890.01	119.92	1982.72	5	108000
S_3	840.00	880.82	15.06	890.56	29	22200
S_4	680.63	680.96	0.96	681.95	29	43000
S_{act}	680.63	680.67	0.026	680.72	30	7400
S_{all}	680.65	680.75	0.056	680.89	30	19400
	Problem G10 (7049.331)					
S_1	7101.50	7346.61	202.15	7682.20	9	147700
S_2	7063.02	8169.68	1866.32	10325.00	3	131600
S_3	7057.27	7406.51	148.60	7518.91	9	148600
S_4	7095.27	7349.83	360.00	7604.39	2	128200
S_5	-	-	-	-	-	-
S_6	-	-	-	-	-	-
S_{act}	7052.30	7199.01	175.01	7943.15	30	42800
S_{all}	7068.04	7141.87	52.27	7239.54	30	9800
	Problem G13 (0.053950)					
S_1	0.053950	0.054908	0.00054	0.055386	6	29800
S_2	0.053950	0.054372	0.00044	0.054968	4	7400
S_3	0.053950	0.054637	0.00017	0.054394	6	7200
S_{act}	0.053950	0.054736	0.001	0.058462	15	19800
	Problem G24 (-5.508013)					
S_1	-5.508013	-5.508013	0.0	-5.508013	30	5800
S_2	-5.508013	-5.508013	0.0	-5.508013	30	24000
S_{act}	-5.508013	-5.508013	0.0	-5.508013	30	21400

on problem $G24$ which has two active inequality constraints where the optimal solution was found for all strategies in each run (see #Fea).

5.2 Comparison with an state-of-the-art algorithm

In this section we compare the best quality results from ACO$_{\mathcal{B}}$ (we use S_{act} as the most efficient search criteria) with respect to the results of a constraint-handling technique representative of the state-of-the-art in the area: Stochastic Ranking (SR) [10]. Table 5 shows for each problem considered, the optimum, and the corresponding Best value found (BF), average (Mean), and Worst values respectively from ACO$_{\mathcal{B}}$ and SR (reported in [10][5]). The performance of ACO$_{\mathcal{B}}$ is comparable in many ways with respect to SR.

From the perspective of the best values (BF) found ACO$_{\mathcal{B}}$ reaches similar values as SR in all the problems considered. For $G02$, ACO$_{\mathcal{B}}$ reached the best known value reported in [5] by using an *ad hoc* boundary operator. On the opposite side, for $G10$, ACO$_{\mathcal{B}}$ did not obtain the optimal solution. However, the results achieved in all cases are highly competitive.

Table 5. *Comparison of ACO$_{\mathcal{B}}$ with respect to a constraint-handling technique representative of the state-of-the-art in the area: stochastic ranking (SR).*

		BF		Mean		Worst	
Prob.	Opt[6]	ACO$_{\mathcal{B}}$	SR	ACO$_{\mathcal{B}}$	SR	ACO$_{\mathcal{B}}$	SR
G01	-15.000	-15.000	-15.000	-15.000	-15.000	-15.000	-15.000
G02	0.803619	0.803619	0.803515	0.802656	0.781975	0.793083	0.726288
G03	1.000	1.000	1.000	1.000	1.000	1.000	1.000
G04	-30665.539	-30665.539	-30665.539	-30665.539	-30665.539	-30666.539	-30665.539
G05	5126.498	5126.50	5126.497	5138.37	5128.881	5132.14	5142.472
G06	-6961.814	-6961.81	-6981.814	-6961.74	-6875.940	-6961.71	-6350.262
G07	24.306	24.37	24.307	24.64	24.374	24.92	24.642
G09	680.630	680.63	680.63	680.67	680.56	680.72	680.763
G10	7049.331	7052.30	7054.316	7199.01	7559.192	7943.15	8835.655
G11	0.75	0.75	0.75	0.75	0.75	0.75	0.75
G13	0.053950	0.053950	0.053957	0.054908	0.057006	0.055386	0.216915
G24	-5.508013	-5.508013	-5.508013	-5.508013	-5.508013	-5.508013	-5.508013
G25	-16.73819	-16.73819	-16.73819	-16.73819 -	16.73819	-16.73819	-16.73819

6 Conclusions and Future Work

In this paper we presented an alterative approach to reach the boundary between the feasible and infeasible search space which could be useful when facing problems with active constraints. For the initial testing of this method we have used an ant colony algorithm as a search engine (ACO$_{\mathcal{B}}$) and a penalty function as a complementary mechanism for problems with more than one constraint. The overall performance of ACO$_{\mathcal{B}}$ was satisfactory for all of the problems considered. The comparison with a state-of-the-art algorithm shows the potential of this method as an alternative or complementary approach for constrained optimization problems. In fact, for some problems, ACO$_{\mathcal{B}}$ was able to improve the respective best known solutions (e.g., $G02$ (with $n = 50$ variables)). It is clear that further improvements should be considered. For example, it is desirable to implement a self-adaptation mechanisms and to try different search engines (e.g., differential evolution and evolution strategies). We also plan to study the performance of our approach in several additional test functions.

---

[5] Except for problems $G24$ and $G25$ for which SR was run by the authors using Thomas Runarsson's code.

Acknowledgments

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. Bilchev, G., Parmee, I.C.: The ant colony metaphor for searching continuous design spaces. Lecture Notes in Computer Science **993** (1995) 25–39
2. Ling, C., Jie, S., Ling, Q., Hongjian, C.: A method for solving optimization problems in continuous space using ant colony algorithm. In Dorigo, M., Caro, G.D., Sampels, M., eds.: Proceedings of the Third International Workshop, (ANTS'2002). Volume 2463 of Lecture Notes in Computer Science. Springer Verlag, (Brussels, Belgium) 288–289
3. Lei, W., Qidi, W.: Further example study on ant system algorithm based continuous space optimization. In: Proceedings of the $4^t h$ Congress on Intelligent and Automation, Shangai, P.R. China (2002) 2541–2545
4. Pourtakdoust, S.H., Nobahari, H.: An extension of ant colony systems to continuos optimization problems. In Dorigo, M., Birattari, M., Blum, C., Gambardella, L.M., Mondada, F., Stützle, T., eds.: Ant Colony Optimization and Swarm Intelligence, 4th International Workshop, ANTS 2004. (Springer-Verlag) 294–301
5. Michalewicz, Z., Nazhiyath, G., Michalewicz, M.: A note on usefulness of geometrical crossover for numerical optimization problems. In et al., L.J.F., ed.: Proceedings of the Fifth Annual Conference on Evolutionary Programming, Cambridge, MA, MIT Press (1996) 305–311
6. Keane, A.: Genetic algorithms digest, v8n16 (1994)
7. Schoenauer, M., Michalewicz, Z.: Evolutionary computation at the edge of feasibility. In Voigt, H.M., Ebeling, W., Rechenberg, I., Schwefel, H.P., eds.: Parallel Problem Solving from Nature – PPSN IV, Berlin, Springer (1996) 245–254
8. Wu, Z., Simpson, A.: A self-adaptive boundary search genetic algorithm and its application to water distribution systems. Journal of Hidraulic Research **40**(2) (2002) 191–203
9. Corne, D., Dorigo, M., Glover, F., eds.: New Ideas in Optimization. McGraw-Hill International (1999)
10. Runarsson, T.P., Yao, X.: Stochastic ranking for constrained evolutionary optimization. IEEE Transactions on Evolutionary Computation **4**(3) (2000) 284–294
11. Hamida, S.B., Schoenauer, M.: ASCHEA: New Results Using Adaptive Segregational Constraint Handling. In: Proceedings of the Congress on Evolutionary Computation 2002 (CEC'2002). Volume 1., Piscataway, New Jersey, IEEE Service Center (2002) 884–889
12. Liang, J., Runarsson, T.P., Mezura-Montes, E., Clerc, M., Suganthan, P.N., Coello Coello, C., Deb, K.: Problem definitions and evaluation criteria for the cec 2006 special session on constrained real-parameter optimization. Technical report, School of Electrical and Electronic Engineering Nanyang Technological University, Singapore, http://www.ntu.edu.sg/home5/lian0012/cec2006/technical_report.pdf (2006)