A Boundary Search based ACO Algorithm Coupled with Stochastic Ranking

Guillermo Leguizamón and Carlos A. Coello Coello

Abstract—In this paper we present a boundary search based ACO algorithm for solving nonlinear constrained optimization problems. The aim of this work is twofold. Firstly, we present a modified search engine which implements a boundary search approach based on a recently proposed ACO metaheuristic for continuous problems. Secondly, we propose the incorporation of the stochastic ranking technique to deal with feasible and infeasible solutions during the search which focuses on the boundary region. In our experimental study we compare the overall performance of the proposed ACO algorithm by including two different complementary constraint-handling techniques: a penalty function and stochastic ranking. In addition, we include in our comparison of results the Stochastic Ranking algorithm, which was originally implemented using an Evolution Strategy as its search engine.

I. 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. However, all of these approaches only deal with unconstrained optimization problems. For example, Ling et al. [13], Lei et al. [10], [11], Dreo et al. [4], Mmonarché et al. [15], and Pourtakdoust et al. [16]. More recently, an extension of the ACO metaheuristic to continuous domains and applied to continuous and mixed discrete-continuous problems is presented by K. Socha [19] which follows the original conception of the ACO approach (see Dorigo and Stützle [3]) in regards of the way the solutions are built, i.e., incrementally. 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 PDF $P_i(x_i)$ 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. In another recent work by Socha et al. [20], the former ideas proposed by Socha [19] regarding continuous domains are extensively presented and details concerning implementation issues are given through the ACO$_R$ algorithm. The experimental study presented by the authors considers a test suite of several unconstrained continuous optimization problems. In addition, an analysis of the behavior of ACO$_R$ is presented regarding the impact of its main parameters on the algorithm's performance: $q$ and $\xi$. In Leguizamón et al. [8] a new constraint-handling technique is implemented in an ACO algorithm for continuous problems based on the former work by Bilchev et al. [1]. Leguizamón et al.'s work introduces a more general boundary approach for solving nonlinear constrained problems which was presented as a possible extension of the ACO algorithms for continuous search spaces. The boundary approach under the ACO metaheuristic showed to be competitive with respect to other state-of-the-art algorithm when dealing with nonlinear problems having active constraints. It is also worth noting that the boundary approach has been studied from the evolutionary computation perspective. For example, Michalewicz et al. [14], Schoenauer et al. [18], and Wu et al. [21]. However, Wu et al.'s work does not involve any explicit boundary operator.

Based on the proposal of Socha et al. [20], Leguizamón et al. [9] adopted the algorithm ACO$_R$. In that work, a new algorithm called ACO$_{SR}(B)$ is compared against ACO$_{SR}(S)$ (see Leguizamón et al. [7]), a boundary search based ACO algorithm designed according the former ideas by Bilchev et al. [1]. The new algorithm ACO$_{SR}(S)$ was found to be a suitable alternative when facing constrained optimization problems. However, both algorithms, ACO$_{SR}(B)$ and ACO$_{SR}(S)$, include a penalty function as their complementary technique adopted to handle the problem's constraints. Although penalty functions are a suitable approach, they usually need an extensive preliminary experimental study to tune the values of their penalty factors. Regardig this situation, we propose in this paper the use of the stochastic ranking approach [17] in order to avoid the use of penalty factors as well as to achieve an improved performance of the ACO algorithm when dealing with constrained problems.

The remainder of this paper is organized as follows. Section II describes the formulation of the general nonlinear optimization problems and some features of these problems that could be exploited when some conditions are met. In addition, a general formulation of the boundary approach (see [8], [7]) is presented. The ACO algorithm ACO$_{SR}(B)$ (based on ACO$_{SR}$) which implements the boundary approach is presented in Section III. On the other hand, Section IV presents ACO$_{SR}(S)$, which is our proposed algorithm for boundary search incorporating stochastic ranking as its complementary constraint-handling technique. In addition, the ACO$_{SR}(P)$ is also presented. The test problems and experimental results are presented and analyzed in Section V. Finally, our conclusions and some possible paths for future

Guillermo Leguizamón is with LIDIC (Research Group), Universidad Nacional de San Luis, Ej. de Los Andes 950 - (D5700HHW) San Luis, ARGENTINA (email: legui@uns.edu.ar). Carlos A. Coello Coello is with CINVESTAV-IPN (Evolutionary Computation Group), Departamento de Computación, Av. IPN No. 2508, Col. San Pedro Zacatenco, Mexico D.F. 07300, MEXICO (email: ccoello@cs.cinvestav.mx).
research are provided in Section VI.

II. THE BOUNDARY SEARCH APPROACH

We avoid solving the general nonlinear programming problem whose aim is to find $x$ so as to optimize:

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

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

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

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

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

At any point $x \in \mathcal{F}$, the constraints $g_k$ that satisfy $g_k(x) = 0$ are the active constraints at $x$. Equality constraints $h_j$ are active at all points of $\mathcal{F}$. It is worth remarking that plenty of problems formulated as above include active constraints at the best known or optimal solutions. For example, for problems with at least one inequality constraint $h_j$, the corresponding optimal solution will lie on the region defined by $h_j(x) = 0$. Furthermore, for many problems, the best solutions may lie on the boundary between the feasible and infeasible search space of some inequality constraints, i.e., the region defined by $g_j(x) = 0$. 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 (accord ending either to the equality and/or inequality constraints) can be a suitable alternative for including in a specific search engine or metaheuristic.

In the following we first explain how the boundary region can be approached given a specific search space; more precisely, the $n$-dimensional space $\mathcal{B}^n$. Then, we also describe the manner in which this search space can be explored assuming a hypothetical search engine and exploration operators. 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.

A. Approaching the boundary

We describe here a general boundary approach (proposed in [8], [7]) which is based on the notion that each point $b$ of the boundary region can be represented by means of two different points $x$ and $y$, where $x$ is some feasible point and $y$ is some infeasible one, i.e., $(x, y)$ can represent one point lying on the boundary by applying a "binary search" on the straight line connecting the points $x$ and $y$ (when considering an equality constraint, $z \in \mathcal{X}$ iff $h(z) \leq 0$; otherwise, $z \in \mathcal{U}$). Figure 1 shows a hypothetical search space including the feasible (shadowed area) and infeasible regions. We can identify four points lying on the boundary $b_1$, $b_2$, $b_3$, and $b_4$ which are respectively obtained from $(x_1, y_1)$, $(x_2, y_2)$, $(x_3, y_3)$, and $(x_4, y_4)$.

![Figure 1](image-url) - Given one feasible and one infeasible point, the corresponding point lying on the boundary can be easily reached by using a simple binary search. In this way, each point on the boundary can be reached from at least a pair of points $(x, y)$ with $x \in \mathcal{F}$ and $y \in \mathcal{U}$.

The binary search applied to each pair of points $(x, y)$ is achieved following the steps described in function BS (see Algorithm 1). For example, a possible application of this process can be seen in Figure 1 where we adopt the pair of points $(x_3, y_3)$ from which we obtain the point $b_3$, 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 $(x_3)$ is moved to point $p_1$. In the next step (labeled (2)) we consider the points $p_1$ and $y_3$ as extreme points for which the mid point is the feasible point $p_2$. Thus, the new feasible point or right extreme of the line is now the point $p_2$. Finally, the last point generated is $b_3$ which can be either lying on or close to the boundary. Condition $(\text{dist to boundary}(m) \leq \delta)$ AND $\text{Feasible}(m)$ defines a threshold to stop the process of approaching the boundary. However, the second part of this condition (i.e., "Feasible(m)") 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 $\delta$ is given by $\log_2(r)$ where $r = (\text{dist}(x, y))/\delta$. Thus, the closer to the boundary, the larger $\log_2(r)$.

B. Exploring the boundary region

So far, we have shown how a point lying on the boundary $b$ can be represented through a pair of points $(x, y)$ with $x \in \mathcal{X}$ and $y \in \mathcal{U}$. Now we need to consider the exploration of the search space where, according to our proposal, can be defined as $\mathcal{G} = \{(x,y) | x \in \mathcal{X} \subset \mathbb{R}^n \text{ and } y \in \mathcal{U} \subset \mathbb{R}^n\}$, that is, the set of pairs of points $(x, y)$ as described above. This space can be considered a genotypic space as
known in evolutionary computation. Since each point from \( G \) represents a point on the boundary, it is necessary the application of the decoder represented by function \( BS \) (see Algorithm 1) to obtain the corresponding phenotype, i.e., the "gene expression" of \((x, y) \in G\). Thus, the set \( B = \{ b \mid b = BS(x, y) \} \) is conformed by the set solutions on the boundary. Each solution in this set is evaluated by function \( B \), which represents a measure of solutions quality and gives as a result an element from the set \( E = \{ e \mid e \in \mathbb{R} \land e = B(b) \} \).

From the above description, it is clear that the search engine must deal with the exploration of space \( G \). For example, from the perspective of evolutionary algorithms, we can create a population of individuals where each one of them represents an element of set \( G \). Therefore, suitable operators to be choosen could be any crossover and/or mutation operators appropriate 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, differential evolution, immune systems, etc. However, from the perspective of the ACO metaheuristic the possibilities are more limited. In this work we will show an alternative for the ACO metaheuristic in the following sections.

C. 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 where the search is always performed "indirectly" on the boundary of the space defined by some of the problem constraints. 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 generating a set of elements of set \( G \). After that, the exploration of \( G \) by the search engine will indirectly and exclusively explore the region defined by \( h(b) \), i.e., all solutions generated will be feasible without requiring any ad-hoc boundary operator.

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. 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. However, a possible search engine could remain focused on a particular constraint over the whole run or may move from one problem constraint to another depending on a particular condition. In our previous work [8], 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 conditions could be considered, for example, taking into account the population diversity or the degree in which some problem constraints are being violated. In this work, as will be explained in a further section, we adopted the parameter \( t_c \) to control the time when the algorithm should focus on a different problem constraint.

III. THE PROPOSED ALGORITHM ACO_{SR}^{B} FOR BOUNDARY APPROACH

In this section we describe the design of the ACO_{SR}^{B} algorithm which implements the boundary search. The search engine involved in ACO_{SR}^{B} is based on the ACO_{B} algorithm presented in [20]. Before explaining the implementation of ACO_{SR}^{B}, we first describe briefly the main characteristics of ACO_{B} as it was proposed and tested in [20] on unconstrained continuous optimization benchmark problems.

![Fig. 2.](image.png)

Fig. 2. (filled line): a continuous probability density function \( p(x_i|\mathbf{s}^p) \) where \( x_i \in [l(i), u(i)] \), and \( \mathbf{s}^p \) is a partial solution under construction (see [20] for further details) and (dotted line): a possible set of three Gaussian functions to achieve by superposition a Gaussian Kernel which approximates the corresponding multimodal Gaussian function (filled line).

Taking into account that the ACO metaheuristic works by incrementally building the solutions according to a biased (by pheromone trail) probabilistic choice of solutions components, the ACO_{SR} algorithm was designed aiming at obtaining a set of \textbf{probability density functions (PDFs)}, each PDF is obtained from the search experience and is used to incrementally build a solution \( x \in \mathbb{R}^n \) considering in turn each component \( x_i \). Figure 2 (filled line) represents a hypothetical PDF that could be eventually found during the search. It can be \textbf{observed} a multimodal PDF used to obtain a value for the variable on dimension \( i \in \mathbb{R}^n \).
{1, ..., n}). To approximate a multimodal PDF that looks like the one in Figure 2, Socha et al. [20] proposed a Gaussian Kernel which is defined as a weighted sum of several one-dimensional Gaussian functions \( g_f(x) \) as follows:

\[
G^i(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{(x-\mu_i)^2}{2}}
\]

where \( i \in \{1, ..., n\} \) identifies the number of dimension, i.e., \( \text{ACO}_B \) uses as many Gaussian kernel PDFs as the number of dimensions of the problem. In addition, \( G^i \) is parameterized with three vectors: \( \omega \), the vector of weights associated with the individual Gaussian functions; \( \mu^i \), the vector of means; and \( \sigma^i \), the vector of standard deviations. All these vectors have cardinality \( k \), which constitutes the number of Gaussian functions involved. Figure 2 (dotted line) shows a superposition of three Gaussian functions which could approximate the hypothetical multimodal Gaussian function (filled line).

In \( \text{ACO}_B \), a solution archive called \( T \) is used to keep track of a number of solutions similarly to the Population Based ACO (PBACO) proposed by Guntensch et al. [6]. The cardinality of archive \( T \) is \( k \), that is, the number of kernels that conform the Gaussian kernel. For each solution \( x_k \in \mathbb{R}^n \), \( \text{ACO}_B \) maintains the corresponding values of each problem dimension, i.e., \( x_1, x_2, ..., x_n \), and the value of the objective function \( f(x_k) \) which are stored satisfying that \( f(x_1) \leq ... \leq f(x_k) \leq ... f(x_n) \). On the other hand, the vector of weights \( \omega \) should satisfy that \( \omega_1 \geq ... \geq \omega_i \geq ... \geq \omega_k \).

The solutions in \( T \) are therefore used to dynamically generate probability density functions involved in the Gaussian kernels. More specifically, for obtaining the Gaussian kernel \( G^i \), the three parameters \( \omega, \mu^i, \) and \( \sigma^i \) need to be calculated. Thus, for each \( G^i \), the values of the i-th variable of the \( k \) solutions in \( T \) become part of the elements of vector \( p^i \), that is, \( \mu^i = \{ p_f, ..., p_{x_k} \} = \{ x_f, ..., x_{x_k} \} \). Vector \( \mu^i \) is generated as follows: each solution that is added to the archive \( T \) is evaluated and ranked (ties are broken randomly). The solutions in \( T \) are stored according to their rank, i.e., the highest the rank of the solution, the lowest the corresponding index in \( T \). The weight \( \omega \) associated to Gaussian function \( g_f^i \) is obtained as:

\[
\omega_i = \frac{1}{2 \sqrt{\pi}} e^{-\frac{(x_i-\mu_i)^2}{2}}
\]

with mean 1.0 and standard deviation \( qk \), where \( q \) is a parameter of \( \text{ACO}_B \) which controls the preference of the ranked solutions. Thus, when \( q \) is small, the best-ranked solutions are preferred, otherwise, a large value for \( q \) implies a more uniform probability. As mentioned in [20], the influence of this parameter on \( \text{ACO}_B \) is similar to adjusting the balance between the iteration-best and the best-so-far pheromone updates used in traditional ACO algorithms. On the other hand, each component of the deviation vector \( \sigma^i = \{ \sigma_1, ..., \sigma_k \} \) is obtained as:

\[
\sigma_i^j = \frac{1}{\sqrt{k}} \sum_{e=1}^{k} \left| x_{i}^j - x_{e}^j \right|
\]

where \( i \in \{1, ..., k\} \) is the kernel number with respect to which the deviation is calculated and \( \xi > 0 \), which is the same for all dimensions, has an effect similar to that of the pheromone evaporation rate in ACO. Thus, the higher the value of \( \xi \), the lower the convergence speed of the algorithm.

For obtaining a solution component at step \( t \) (in the construction solution process) it is only necessary to calculate the \( i \)-th component of \( \sigma^i \) since the sampling process of Gaussian kernel \( G^i \) is accomplished as follows. Given the elements of vector \( \omega \) calculated as in Eq. 2, the sampling is done in two phases: 1) choose one of the \( k \) Gaussian functions of \( G^i \) according to the following probability:

\[
p_i = \frac{\omega_i}{\sum_{e=1}^{k} \omega_e}.
\]

and, 2) after function \( g_f^i \) has been chosen, a sampling is accomplished (perhaps using a random number generator based on a parameterized normal or an uniform distribution in conjugation with, for instance, the Box-Muller method [21]). Since at each step only one Gaussian function is used (let us say \( g_f^i \)), it is only needed \( \sigma_i^j \) instead of the whole vector \( \sigma^i \). The pheromone update is achieved by considering a set \( \mathcal{A} \) of the newly generated solutions. The new \( T \) (in the next algorithm iteration) is obtained as \( T = rank(T \oplus A) \), i.e., the old solutions in the archive \( T \) plus the set of newly created solutions \( A \) are ranked. In other words, the old solutions compete against the newly generated ones to conform the updated \( T \) which maintains its cardinality \( k \) throughout the whole search process.

To adapt \( \text{ACO}_B \) to deal with constrained problems by implementing the boundary approach described above is rather straightforward. The proposed algorithm \( \text{ACO}_B \) instead of maintaining one archive \( T \), now maintains two archives for similar purposes, \( T_{FB} \) and \( T_{U} \) which represent respectively the points on the feasible and infeasible parts of space \( \mathcal{G} \). A third archive, \( T_{B} \), is also considered which is obtained by applying function \( BS \) to each point from \( T_{FB} \) and \( T_{U} \). More precisely, \( T_{B} = \{ b_x, b_y, BS(x, y) \} e = 1, ..., k \). Solutions in \( T_{B} \) are evaluated by means of function \( BS \). It is worth remarking that solutions in \( T_{B} \) are ranked according to the solution quality given by \( BS \). Taking into account this ranking, the solutions in \( T_{FB} \) and \( T_{U} \) are then ranked accordingly.

As in the original \( \text{ACO}_B \) algorithm, vector \( w \) is intended for sampling the chosen Gaussian function, however, the situation is different in \( \text{ACO}_B \) since there exist two independent archives \( T_{FB} \) and \( T_{U} \) from which the Gaussian Kernels are built, i.e., to explore the search space \( \mathcal{G} \), it is necessary to process both archives from which the solutions on the boundary are obtained. In addition, we define two additional structures \( A_{FB} \) and \( A_{U} \) associated respectively to archives \( T_{FB} \) and \( T_{U} \). These two structures, similarly as

\[\text{Set } A \text{ represents the set of ants according to Socha et al. [20].} \]
TABLE I

REPRESENTATION OF THE ACOR_{BR} SEARCH SPACE DIVIDED IN FEASIBLE AND INFEASIBLE POINTS

in the original ACOR, represent the newly solutions found according to the Gaussian kernels from T_{F} and T_{U}. Table I represents a general outline of the archives T_{F}, T_{U}, T_{B}, w, and E. The last one is associated to T_{B} and maintains the value corresponding to the evaluation quality of solution in T_{B}. It should be noticed that T_{B} is not used to build any Gaussian Kernel, however, the ranking of the solution in it will influence the ranking of solutions in T_{F} and T_{U}, which clearly influences the generation of new and better quality solutions in the space G.

A general outline of ACOR_{BR} is presented in Algorithm 2 which displays its main components. In line 1, archives T_{F} and T_{U} are initialized by randomly generated solutions in the feasible and infeasible search space regarding the problem constraint at hand. Similarly, vector w is initialized according to Eq. 2 which includes the parameters q and k as explained above. The main loop includes a call to function “Boundary”, which in charge of applying function BS to each pair of points respectively from T_{F} and T_{U} and returns the archive T_{B}. Then, function “BuildSols” is in charge of generating new solutions through the Gaussian kernel obtained from the corresponding archives (lines 4 and 5). In order to further obtain A, i.e., the newly generated solutions on the boundary, function “Boundary” is then applied to A_{F} and A_{U}. After that, T_{B} plus A_{B} are ranked according to the solutions quality given by function \phi, and the best first k solutions in the ranking will be now part of the archive T_{B} which is used as a reference to get the new T_{F} and T_{U}.

Let us say that the new set of points on the boundary is T_{B} = \{b_{1}, \ldots, b_{k}\} where b_{i} either comes from T_{B} or A_{B}, therefore the new T_{F} and T_{U} are obtained respectively from T_{F} \oplus A_{F} and T_{U} \oplus A_{U} taking into account the ranked solutions in the new T_{B}. This is precisely what the function “Update” does.

Algorithm 2 A general outline of the ACOR_{BR} algorithm

IV. THE ACOR_{Pen} AND ACOR_{ER} ALGORITHMS

Based on the above modifications for the original ACOR, we define here two algorithms, ACOR_{Pen} and ACOR_{ER}. The name ACOR_{Pen} corresponds to the algorithm called ACOR_{ER} which was proposed and studied in [9]. The second algorithm, which constitutes the main proposal of this work, is called here ACOR_{ER} where the complementary constraint-handling technique is the stochastic ranking approach proposed by Rummanson et al. [17]. The main characteristic of ACOR_{Pen} is that the function \phi used to find the values in the structure E (see Table I) must include a penalty factor in order to evaluate the solutions on the boundary (structure T_{B}). However, it is well known that the main drawback

2 We change the name here since both algorithms are based on Socha et al’s proposal where the difference is in the complementary constraint-handling technique.
of this technique is the problem to find the most suitable penalty function and/or the corresponding penalty factors involved. In Section V, we will show the penalty function used and the corresponding penalty factors which have been extensively studied in Leguizamón et al. [8], [7], and [9].

For the $ACO_{ES}$ algorithm, the mechanism is slightly different, nevertheless, its implementation is straightforward. First of all, it is necessary to include another structure associated to archive $T_B$ to keep the extent of violation of the problem constraints. Let us call this new structure $V = \{v(b_1), \ldots, v(b_j), \ldots, v(b_k)\}$, where $v$ is a function that returns precisely the extent of violations of the problem constraints given by $v(b) = \sum_{i=1}^{q} \max(0, g_j(b))^2 + \sum_{j=q+1}^{l} |h_j(b)|^2$ (similarly as defined in [17], however, any other suitable function can be applied) and $\phi = f$, i.e., the objective function. After that, function "Sort" (line 7, Algorithm 2) should be accordingly changed. Following the proposal of Runarsson et al., the former function "Sort" which implements any classical sorting algorithm, is modified now in the way that implements a sort-like procedure (see Algorithm 3) to proceed with the stochastic ranking of the newly generated solutions. It should be noticed that

Algorithm 3 A general outline of the stochastic ranking algorithm uses a bubble-sort like algorithm as defined in [17]. $P_f$ 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 $\lambda$ represent respectively the maximum number of sweeps and number of solutions that are ranked by comparing adjacent solutions in at least $\lambda$ sweeps, and $\text{rand} \in U(0,1)$.

1: $I_j = j \forall j \in \{1, \ldots, \lambda\}$
2: for $j$ in $1 : N$ do
3: for $j$ in $1 : \lambda - 1$ do
4: if $(v(x_{I_j}) \geq v(x_{I_{j+1}}))((\text{rand} < P_f))$ then
5: if $(v(x_{I_j}) > v(x_{I_{j+1}}))$ then
6: swap$(I_j, I_{j+1})$
7: end if
8: else
9: if $(v(x_{I_j}) > v(x_{I_{j+1}}))$ then
10: swap$(I_j, I_{j+1})$
11: end if
12: end if
13: end for
14: if no swap done then
15: break
16: end if
17: end for

the indexes $I_j$ and $I_{j+1}$ in function "swap" point to the corresponding structures (e.g. $T_B$ or other) to produce the swaps when necessary. Runarsson et al. suggest the setting $N = \lambda$ for the number of solutions adjacent to be compared. In our case, $\lambda$ indicated the number of solutions in the corresponding structure to be sorted. Thus, if $|T_B| = k$ and $|\text{alg}| = Na$, then $\lambda = k + Na$ (see line 8 in Algorithm 3).

V. EXPERIMENTS AND RESULTS

The main objective of our experimental study is to analyze the quality of results as well as the performance of $ACO_{ES}^{(Pen)}$ and $ACO_{ES}^{(SR)}$ regarding the number of feasible solutions found. In addition, we make a comparison with one of these two algorithms and the original stochastic ranking approach, as described in [17] (using an evolution strategy as its search engine). Before presenting the results we will describe some common characteristics of $ACO_{ES}^{(Pen)}$ and $ACO_{ES}^{(SR)}$ regarding their application to the different test cases. Indeed, $ACO_{ES}^{(Pen)}$ and $ACO_{ES}^{(SR)}$ require minimum changes when applied to the different test cases considered: the objective function, number of variables, range of each variable, and constraints. However, the policy to determine on which constraint the search should focus needs to be considered when a problem has 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 policies to deal with the way of approaching to the boundary were extensively studied in Leguizamón et al. [8], [7] for the algorithm $ACO_{ES}^{(SR)}$. From these earlier results, we adopt the so called $S_{act}$ policy, which showed the best performance in all the test cases studied. However, the other policies are also a valuable and efficient alternative when no information is available with respect to the possible active constraints. In our experiments, the condition to produce a change on the search from one constraint to another is given by an elapsed number of iterations and is represented by the parameter $t_c$ as explained in Section II-C. In addition, for problems with more than one constraint, we incorporate a penalty function for algorithm $ACO_{ES}^{(Pen)}$ of the form:

$$\phi(x,\mu) = f(x) + \mu \sum_{j=1}^{q} \max(0, g_j(x)) + \sum_{j=q+1}^{m} |h_j(x)| (5)$$

where $\mu$ is a fixed penalty factor. Also, it is worth remarking that each solution is always lying on the boundary of the feasible space corresponding to the constraint under consideration. This sort of penalty function was previously adopted in [8], [7] due to its simplicity, since our interest was to assess the advantages of the boundary approach proposed. However, other constraint-handling techniques are evidently possible as the stochastic ranking approach proposed in this article (algorithm $ACO_{ES}^{(SR)}$). The penalty factors $\mu$ used in $ACO_{ES}^{(Pen)}$ were experimentally determined for each particular problem (see [9]) and are showed later. All the algorithms considered in this experimental study (i.e., $ACO_{ES}^{(Pen)}$, $ACO_{ES}^{(SR)}$, and SR) were 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 G07, G09, G10, G11, G13, G14, G15, G17, G21, G23, G24 [12], and G25 [5]. At earlier experiments with ACO_{Pen} in [9], we initially chose similar parameter settings as those used in [20] where \( N_a = 2, k = 50, \xi = 0.85 \), and \( q \in [0.0001, 0.01] \); where the higher value for parameter \( q \) was chosen for multimodal functions. The preliminary results from ACO_{Pen} by using the above parameter setting was rather discouraging since the algorithm was not capable of achieving any feasible solution for all the test problems adopted. After that, we considered a larger number of ants (i.e., \( N_a \gg 2 \)) for generating a larger sampling of solutions according to the \( \text{Gaussian kernels. More specifically, we set } N_a = 50 \) which was the setting for the number of ants used in ACO_{SR} and ACO_{SR}^P in the experiments presented in this section. The penalty factors involved in function \( \phi \) (Eq. 5) for each problem using ACO_{Pen} were as follows: G01 (\( \mu = 1000 \)), G04 (\( \mu = 5000000 \)), G05 (\( \mu = 10 \)), G06 (\( \mu = 0.1 \)), G07 (\( \mu = 20000 \)), G09 (\( \mu = 2000000 \)), G10 (\( \mu = 20000000 \)), G13 (\( \mu = 0.1 \)), G14 (\( \mu = 150 \)), G15 (\( \mu = 10 \)), G17 (\( \mu = 1000 \)), G21 (\( \mu = 3000 \)), G23 (\( \mu = 10000 \)), and G24 (\( \mu = 100000 \)). All of these values were set based on our previous work [3] in which similar values were adopted for the so called ACO_{SR}. On the other hand, for ACO_{SR} we set \( \Pi = 0.45 \) and \( N = \lambda \). The whole experimental study was performed on a Laptop with an Intel® Pentium® M Processor 725, running at 1.6 GHz, and with 512 Mbytes of RAM. The ACO_{SR} algorithm was implemented in C Language running under Suse-Linux. It is important to remark that the test suite considered includes problems with only one constraint. For these problems (G02, G03, G11, and G25), the application of either ACO_{SR} or ACO_{SR}^P gives the same results. The reason is because when a problem has only one constraint, the boundary approach generates only feasible solutions, i.e.,

there is no need to use a complementary constraint-handling technique. Thus, both ACO approaches still have the same performance, because their search engine is exactly the same. However, we will further show the results for these problems when comparing the performance of ACO_{SR}^P with SR.

Table II displays the results from algorithms ACO_{SR} and ACO_{SR}^P for the test problems with more than one constraint. The columns show respectively the best found (BF), mean (Mean), and worst (Worst) values and the number of feasible solutions found out of 30 independent runs. The (*) in column BF means that the algorithm achieved the best known or optimal value whereas (+) means the best found value is very close to the best known or optimal value. In column BF some values are in boldface indicating that the corresponding algorithm found the best value. It can be observed that for G01, G04, G06, G07, G09, G13, G15, and G25 the two algorithms behave very similarly regarding quality of solutions. However, the number of feasible solutions found is always 30 for ACO_{SR}. On the other hand, we can observe that ACO_{SR}^P achieved a better performance for problems G05, G10, G14, G17, G21, and G23, considering both, the quality of results and number of feasible solutions found. It is important to remark that for problem G23, we used \( \Pi = 0.2 \) in order to obtain feasible solutions, otherwise (with \( \Pi = 0.45 \)), ACO_{SR}^P was not able to find any feasible solutions. Finally, Table III shows the results from ACO_{SR} and SR for some problems. The remaining problems considered are not shown since both algorithms perform similarly, including those problems with only one constraint. However, for problems G02, G10, G14, G17, G21, and G24 these two algorithms behave differently. For G02, ACO_{SR} found the best known value for this problem, however, for G10, SR found a slightly better value than ACO_{SR}. For problems G14, G21, and G23, the results shows a more clear difference between these two algorithms where ACO_{SR}^P clearly outperforms SR (it must be noticed that we set for ACO_{SR}^P and SR, \( \Pi = 0.45 \) except for G23).
for which $P_f = 0.2$)

VI. DISCUSSION

In this paper, we presented an alternative ACO algorithm $(\text{ACO}^{\text{SR}})$ with a new search engine for implementing the boundary search approach. The search engine is an adaptation of a recent proposal for continuous problems $(\text{ACO}_k)$. The new algorithm, called $(\text{ACO}^{\text{SR}})$, includes stochastic ranking as a complementary mechanism for problems with more than one constraint. For testing $(\text{ACO}^{\text{SR}})$, we have also used a version with a penalty function $(\text{ACO}^{\text{Pen}})$. The results showed a better performance of $(\text{ACO}^{\text{Pen}})$ with respect to $(\text{ACO}^{\text{SR}})$, especially regarding the number of feasible solutions. In addition, the overall performance of $(\text{ACO}^{\text{SR}})$ was compared to $(\text{ACO}_k)$, showing the potential of this method as an alternative or complementary approach for constrained optimization problems. Future work involves the use of a hybrid version of $(\text{ACO}^{\text{SR}})$ with local search, e.g., by doing the main exploration on $Q$ and a complementary exploration on $B$. In addition, we are also interested in the design of a more general approach which includes the boundary approach as a component that can be triggered when certain conditions are met. Finally, a deeper analysis is necessary for the boundary approach as proposed here when facing constrained problems with a complicated feasible region.

ACKNOWLEDGMENT

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 gratefully acknowledges support from CONACYT project no. 42435-Y.

REFERENCES


