The Ant Colony Metaphor in Continuous Spaces using Boundary Search

Guillermo Leguizamón

Laboratorio de Investigación en Inteligencia Computacional
Universidad Nacional de San Luis
Ejército de Los Andes 950
(5700) San luis, Argentina
legui@unsl.edu.ar

Abstract

This paper presents an application of the ant colony metaphor for continuous space optimization problems. The ant algortihm proposed works following the principle of the ant colony approach, i.e., a population of agents iteratively, cooperatively, and independently search for a solution. Each ant in the distributed algorithm applies a local search operator which explores the neighborhood region of a particular point in the search space (*individual search level*). The local search operator is designed for exploring the boundary between the feasible and infeasible search space. On the other hand, each ant obtains global information from the colony in order to exploit the more promising regions of the search space (*cooperation level*). The ant colony based algorithm presented here was successfully applied to two widely studied and interesting constrained numerical optimization test cases.

keywords: *ant colony optimization, evolutionary algorithms, constraint optimization problems, boundary search.*

1 Introduction

The general nonlinear programming (NLP) problem is to find $\mathbf{x}$ so as to

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

where $\mathbf{x} \in \mathcal{F} \subseteq \mathcal{S}$. The set $\mathcal{S} \subseteq \mathbb{R}^n$ defines the search space and the set $\mathcal{F} \subseteq \mathcal{S}$ defines a *feasible* search space. 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):

$$l(i) \leq x_i \leq u(i) \qquad 1 \leq i \leq n$$

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

$$g_i(\mathbf{x}) \leq 0 \quad \text{for} \quad j = 1, ..., q \qquad\qquad h_j(\mathbf{x}) = 0 \quad \text{for} \quad j = q+1, ..., m$$

At any point $\mathbf{x} \in \mathcal{F}$, the constraints g_i that satisfy $g_j(\mathbf{x}) = 0$ are called the *active* constraints at $\mathbf{x}$.

In the last few years evolutionary algorithms and ant colony optimization have been studied and found application to numerical optimization problems. Evolutionary techniques have been widely used for solving constrained optimization problems for which several search operators have been proposed and investigated. However some difficulties emerge when the evolutionary algorithm is not able of searching the boundary areas of the feasible and infeasible search space [10]. A novel approach to avoid the above obstacle has been investigated in [11, 13, 14]. That approach which belongs to the category of the methods based in preserving the feasibility of the solutions, incorporates operators that search the boundary of the feasible and infeasible regions in an efficient way.

On the other hand, some few experiences on applying the ant colony optimization (ACO) model to numerical optimization problems can be found in the literature and they are not as developed as evolutionary techniques. The ant colony optimization model [1, 2] was initially developed for *order based* problems (e.g., the Traveling Salesman Problem, Quadratic Assignment Problem). Recent developments [3, 4, 5, 8] include the application of the ACO model to *non ordering* or *subset* problems (e.g., Multiple Knapsack Problem) and continuous search space. For solving subset problems [8], the ACO approach is applied by moving the *trail* from the problems' connections to the problems' components. On the other hand, problems involving continuous variables and constraints were studied in [3] where a discrete structure is considered which represents a finite set a possible directions to direct the individual search. Thus, following one of the main ideas of the ACO approach regarding global information, the information *trail* is laid on directions. Therefore, the agents move from the nest (a particular set of points) to the more profitable regions in the search space taking into account the *trail* information.

This paper shows the applicability of the ACO approach for solving constrained numerical optimization problems by combining together in one algorithm the features of the ACO approach and the boundary operators formerly included in evolutionary algorithms —i.e., operators which consider the boundary of the feasible and infeasible search space. The design of the ACO algorithm is based on different works on Evolutionary Algorithms (EAs) [11, 13] incorporating special genetic operators (mutation and crossover) to search on the boundary of the feasible search space and taking into account the active constrains of the problem.

In addition to the global information to decide which direction to follow, each agent in the ant algorithm presented here uses a search operator similar to those mutation operators defined in [11, 13, 14]. Additionally two approaches are tested with respect to the neighborhood size of a particular point in the search space.

The paper is organized as follows. The following section describes the main characteristics of the ACO approach as well as its formulation for solving continuous space problems. Section 3 briefly describes the main ideas behind boundary operators in evolutionary algorithms and the two test cases considered in our experiments. Section 4 presents the ACO approach as a general technique for searching on the boundary as well as two version of an ACO algorithm for NLP problems. Section 5 describes the experiments and results. The last section discusses the more relevant results regarding the two ACO algorithms implemented.

2 The Ant Colony Optimization for Continuous Spaces

Ant algorithms are multi-agent systems in which the behavior of each simple agent, called *artificial ant* of *ant*, is inspired by the behavior of real ants. The ant colony optimization (ACO) meta-heuristic defines a particular class of ant algorithms. Essentially, these algorithms work by matching the notion

of candidate solution with the route taken by an ant between two (possibly the same) places. Ants leave a *trail* as they travel, and routes which correspond to good solutions will get a stronger trail than routes which lead to poor solutions. By this way, the trail strength will affect the route taken by future ants, i.e. previously generated solutions (routes taken by past ants) affect (via trail strength) the solutions generated by future ants.
The traveling salesman problem (TSP) plays an important role in the ant colony optimization because it was the first problem to be attacked by considering the above principles concerning the behavior or real ants. An extensive source of information related to ACO meta-heuristic and its applications can be found in [6].
Another interesting application of the ACO approach is concerned with numerical optimization [3, 5]. In order to apply an ACO algorithm an strategy is designed for modeling a continuous nest neighborhood with a discrete structure as appearing in the ACO model either for ordering or subset problems. That strategy determines a set $\{d_1, d_2, ..., d_k\}$, where k (number of directions) is a parameter of the method ($k = 4$ in Fig. 1). Each d_i is represented as a vector (or point) in a n-dimensional search space.

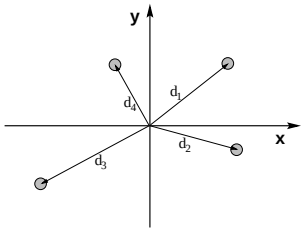

Figure 1: Nest with four points

The structure of the ACO algorithm [3] is showed in Figure 2. Initially, the nest structure is initialized by generating uniformly random starting vectors which represent search directions as appear in Figure 1. Next, a value R_i (for $1 \leq i \leq n$) is defined in order to establish a search radius around each dimension of the points belonging to the nest. R_i values are subject to the bounds of each variable of the problem. The search radius determines the maximum extent of the subspace (neighborhood) to be considered in each *cycle* of the algorithm. Thus, `initialize A(t)` allocates ants on various directions; `evaluate A(t)` is a call to the objective function for all ants; `update_trail A(t)` is in charge of, proportionally, add a trail quantity to the particular directions the ants have selected according to their fitness. Then, `allocate_ants A(t)` allocates the ants by selecting directions using a Roulette Wheel selection on the trail quantity and making a random step from the location of the best previous ants that have selected the same direction and finally, `evaporate A(t)` decrements the trail on each direction. The random step can be implemented as

$$\Delta_{R_i}(t) = R_i.(1 - r^{(1-t/T)^b}) \tag{1}$$

where R_i is a problem dependent fixed parameter; r is a random number from $[0..1]$; T is the maximal cycle number, and b is a system parameter determining the degree of non-uniformity. $\Delta_{R_i}(t)$ returns a value in the range $[0..R_i]$ such that the probability of $\Delta_{R_i}(t)$ being close to 0 increases as t increases. Figure 3 represents the evolution of a two dimensional vector $p_i(t)$ on direction i for $t \in \{0, 1, 2, 3, 4\}$. For simplicity let's assume that $R_1 = R_2 = R$. Thus, a n-dimensional rectangle represents the neighborhood of a particular point. According to the structure of the ACO algorithm, at the first cycle, a number of ants are allocated on the k directions—i.e.; the ants that were allocated on direction i at

```
procedure ACO Algorithm
begin
    t = 0
    initialize A(t)
    evaluate A(t)
    while (not termination_condition ) do
    begin
        t = t + 1
        update_trail A(t)
        send_ants A(t)
        evaluate A(t)
        evaporate A(t)
    end
end
```

Figure 2: Structure of the ACO algorithm for continuous problems

time 0 will start the search from vector $p_i(0)$. In Figure 3, $p_i(0)$ is the starting vector on direction i that belong to the nest structure, $p_i(1)$ is the best point found by the ants allocated on direction i regarding the search radius $\Delta_R(0)$[1]. As t get increased, new regions of the search space are explored: $p_i(2)$, $p_i(3)$, $p_i(4)$ and so on. During the run if certain direction do not result in any improvement, they do no participate in the trail adding process and the reverse (evaporation) process diverts attention away from them. This can be thought of as an analogy of a food source exhausting.

This ant colony model comprises at least two main levels of abstractions:

1. *individual search*: describes the employed individual search strategy—i.e., stochastic hill-climbing, steepest descent, line search, etc.

2. *cooperation*: involves the information interchange among the agents which consists of a joint search effort towards certain directions. That information is represented by τ, where τ_i is the

[1]Note that $\Delta_R(t)$ is not a monotonically decreasing value (See Eq. 1)

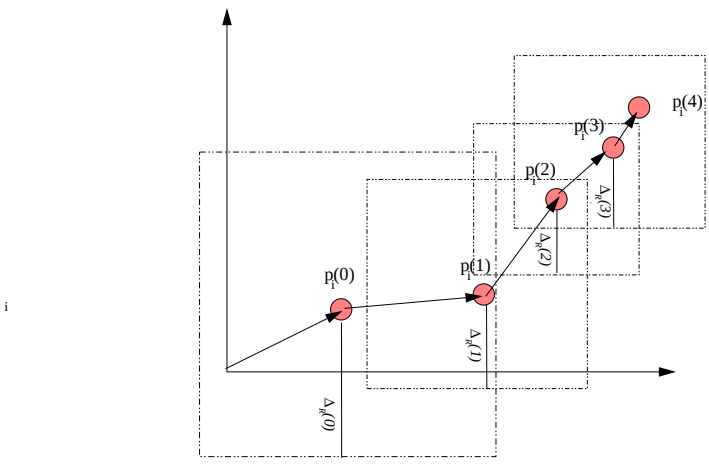

Figure 3: Evolved points and their respective neighborhood area

amount of trail laid on direction i. During the search, before starting the individual local search, the agents select randomly a direction to search with probability:

$$P_i(t) = \frac{[\tau_i(t)]^\alpha \cdot [\eta_i(t)]^\beta}{\sum_k [\tau_k(t)]^\alpha \cdot [\eta_k(t)]^\beta} \quad (2)$$

η_i is either 1 (as used in our experiments) or incorporates some problem knowledge[1], and α and β define a trade-off between the ant colony dynamic and heuristic value.

Related to the variation of the amount of the trail on each direction (τ_i), the ACO algorithm update τ_i by (1) $\tau_i = \tau_i + \Delta\tau_i$ in `update_trail A(t)` and (2) $\tau_i = \rho \cdot \tau_i$ in `evaporate_trail A(t)`, where $\Delta\tau_i$ is a quantity proportional to the fitness of the best ant on direction i and $(1 - \rho)$ is the coefficient of evaporation.

3 Boundary Search Operators in Evolutionary Algorithms

Evolutionary algorithms for solving general constrained optimization problems involve some type of constraint handling techniques regarding the feasible and infeasible search space [9]. These techniques are concerned with the potential generation of infeasible solutions during the search. Some approaches include several instances of penalization and repair algorithms. Alternatively it is possible to use decoders in order to drive the search inside a feasible search space —i.e., the decoder search space. More recently a method called *strategic oscillation* [9, 12] was developed in conjunction with the evolutionary strategy *scatter search*. This method is a basic strategy for exploring the boundary between feasible and infeasible parts of the search space. It proceeds as follows: approaches and crosses the feasibility boundary by a design that is implemented either by adaptive penalties and "inducements". In the context of constrained parameter optimization, an analogous approach was proposed in evolutionary algorithms by applying special operators designed for searching the boundary between the feasible search space [11, 13, 14]. A useful property of the boundary search space is that it is closed under the application of the boundary operators. Additionally it is worth remarking that the initial population is only composed of individuals laying on the boundary search space. Therefore all the potential individuals to be generated will lay on the boundary as well. In regards of this boundary approach, we describe in the following two examples of numerical constrained optimization problems for which evolutionary techniques were applied.
The first test case is an interesting constrained numerical optimization problem proposed by Keane [7] which consists in maximizing the function:

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

with $\prod_{i=1}^n x_i \geq 0.75$, $\sum_{i=1}^n \leq 7.5n$, and $0 \leq x_i \leq 10$ for $1 \leq i \leq n$.
Function F_K is nonlinear and its global maximum is unknown, lying somewhere near the origin. The problem has one non linear constraint and one linear constraint; the last one is inactive around the origin.

In the following some of the main components of the evolutionary algorithm designed for searching on the boundary feasible search space are briefly described.

[1] For some continuous domains gradient information could be used if available.

- Initialization: each solution $\mathbf{x} = (x_1, ..., x_n)$ in the initial population is obtained as follows: randomly choose a positive variable for x_i and use its inverse as a variable for x_{i+1}. The last variable is either 0.75 (when n is odd), or is multiplied by 0.75 (if n is even).

- Crossover: select two parents $\mathbf{x}$ and $\mathbf{y}$. The offspring $\mathbf{z}$ is obtained by:

$$z_i = x_i^{\alpha} y_i^{(1-\alpha)} \quad i \in [1, n], \text{ with } \alpha \text{ randomly chosen in } [0, 1]$$

  This operator is called *geometrical crossover*.

- Mutation: pick two variables randomly and then multiplying one by a random factor q and the other by $\frac{1}{q}$ (restrict q respect to the bounds on the variables).

The second test case [11] correspond to the following optimization problem:

$$F_S(\mathbf{x}) = (\sqrt{n})^n \cdot \prod_{i=1}^{n} x_i \,,$$

where $\sum_{i=1}^{n} x_i^2 = 1$ and $0 \leq x_i \leq 1$ for $1 \leq i \leq n$. The function has a global solution at $(x_1, x_2, ..., x_n) = (\frac{1}{\sqrt{n}}, ..., \frac{1}{\sqrt{n}})$ and the value of the function on this point is 1.

In the following some of the main components of the evolutionary algorithm designed for searching on the boundary feasible search space for F_S are briefly described.

- Initialization: each solution $\mathbf{x} = (x_1, ..., x_n)$ of the initial population is obtained by generating n variables y_i and calculating $s = \sqrt{\sum_{i=1}^{n} y_i^2}$. Then $x_i = y_i/s$ for $i \in [1, n]$.

- Crossover: select two parents $\mathbf{x}$ and $\mathbf{y}$. The offspring $\mathbf{z}$ is obtained by:

$$z_i = \sqrt{\alpha x_i^2 + (1-\alpha) y_i^2} \quad i \in [1, n], \text{ with } \alpha \text{ randomly chosen in } [0, 1]$$

  This operator is called *sphere crossover*.

- Mutation: pick two different variables randomly from solution $\mathbf{x}$. Let's say x_i and x_j and a random number $p \in (0, 1)$

$$x_i \to p \cdot x_i \quad \text{and} \quad x_j \to q \cdot x_j \quad \text{where} \quad q = \sqrt{(\tfrac{x_i}{x_j})^2 (1 - p^2) + 1}$$

4 Boundary Search Operators in the ACO Algorithm

It is common situation for many constrained optimization problems that some constraints are active at the target global optimum and this optimum lies on the boundary of the feasible search space. Our proposal towards to redesign some components of the ACO algorithm for continuous spaces as was presented in Section 2 in order to guide the search around the boundary of the feasible and infeasible search space. In this new version of the ACO algorithm the nest must consist of a set of points either very close the boundary or just on the boundary where some or all constraints of the problem are active. However, the way in which we obtain those points could be strongly determined by the problem at hand as well as the way the algorithm will explore the region in order to keep the new generated points as close as possible to the boundary during the individual search stage. Therefore, an ACO

algorithm designed for searching on the boundary search space have to include an appropriate process of generation of the initial points as well as a special individual search technique—i.e., random step (Eq. 1) must be changed accordingly. Recent experiments from the community of evolutionary computation showed outstanding results [11, 13, 14] by applying evolutionary algorithms which restricted the search of the solutions to the boundary of the feasible part of the search space. In order to fulfill that requirement some special evolution operators were designed such as the geometrical and sphere crossovers and two specific mutation operators. In reference to our goal we design a new version of the ACO algorithm for solving the two test cases F_K and F_S described in Section 3. This new version of the ACO algorithm based in earlier works on boundary search [11, 13, 14] satisfies the requirements above mentioned concerning the initialization process and individual search.
In addition to the ACO algorithm based in boundary operators, we also implemented an ACO algorithm based in the approach proposed in [5] for constrained optimization in order to compare our proposal with an ACO algorithm without using any boundary operator. In the following, their respective implementations will be referred as AC0-B and ACO-nonB.

4.1 ACO-nonB algorithm

Related to this article it is important to consider the ACO algorithm [5] for constrained optimization problems in which is included a constraint handling technique based on the distance of a solution from the feasible search space —i.e., the smaller the distance, the more acceptable the solution is. The acceptability of constraint violation is implemented as a linear function of the ratio of the feasible region to the overall search space.
The design of ACO-nonB algorithm is based on the ACO algorithm described in [5]. In the following the main characteristics concerning the handling constraints approach are explained.
For the two test cases considered in this paper ACO-nonB algorithm implements (at the *individual search* level) a method based in *penalty functions* which takes into account the violation of constraints. Thus, for test case F_K, an infeasible solution is penalized by decreasing the objective value $F_K(\mathbf{x})$ by a rate $p = 0.75/(0.75 - \prod_{i=1}^{n} x_i)$. Therefore, the smaller the value ($\prod_{i=1}^{n} x_i$), the bigger the penalization applied to the solution.
On the other hand, for test case F_S, the penalization method includes *dead penalization* for that solutions which $|1 - \sum_{i=1}^{n} x_i^2| > \frac{I_a}{2}$. Otherwise, the solution is penalized by decreasing the objective value $F_S(\mathbf{x})$ by a rate $|1 - \sum_{i=1}^{n} x_i^2|/I_a$. The parameter I_a determines the interval of acceptable solutions with respect to their objective value. The initial value of I_a is established experimentally and is decreased as a function of the elapsed cycles of the algorithm. At the end of the running I_a is close to zero —i.e., only feasible solutions are accepted at this stage. In our experiments, I_a was set to 1.
For both test cases considered, the initialization process proceeds in a similar way. The nest is a set of points randomly generated subject to the interval of the respective problem variables. Thus, for function F_K; $x_i \in [0, 10]$ and, for function F_S; $x_i \in [0, 1]$.

4.2 ACO-B algorithm

The ACO-B algorithm for F_K proceeds as follows. Before sending the ants on different directions, `initialize A(t)` determines the nest in the search space. That nest is a set of points which lays on the boundary of the feasible search space. The process to obtain the nest (the initial solutions) is the same as the process used in the evolutionary algorithm described in section 3 corresponding to function F_K. Randomly choose a positive variable for x_i and use its inverse as a variable for x_{i+1},

the last variable is either 0.75 (when n is odd), or is multiplied by 0.75 (if n is even). The *individual search* level for function F_K is implemented as follows: the parameters $R_i = R = 10$ for $i = 1, .., n$ since $0 \leq x_i \leq 10$, therefore $\Delta_R(t) \in [0, 10]$. Given a solution $\mathbf{x} = (x_1, ..., x_i, ..., x_j, ..., x_n)$, we randomly choose two variables i and j. Let's assume that $x_i < x_j$ (for $x_i \geq x_j$ we proceed in a similar way). Next we obtain a value $q \in (\max\{q_i, q_j\}, 1)$ where q_i and q_j are chosen such as they satisfy the following:

$$\frac{x_j}{q_j} = V_j \text{ and } V_j = \min\{x_j + \Delta_R(t), 10\} \tag{3}$$

$$x_i \cdot q_i = V_i \text{ and } V_i = \max\{0, x_i - \Delta_R(t)\} \tag{4}$$

Thus, the new solution, which also lies on the boundary, is obtained as:

$$\mathbf{x}' = (x_1, ..., x_i \cdot q, ..., \frac{x_j}{q}, ..., x_n)$$

where $x_i \cdot q \geq V_i$ and $\frac{x_j}{q} \leq V_j$.
For the test case F_S the ACO-B algorithm proceeds as before, except that the *nest* is determined by generating a set of points where each one of them is obtained following the initialization process described in Section 3 regarding the evolutionary algorithm for this function.
The *individual search level* for function F_S is implemented as follows: the parameters $R_i = R = 1$ for $i = 1, .., n$ since $0 \leq x_i \leq 1$, therefore $\Delta_R(t) \in [0, 1]$. Given a solution $\mathbf{x} = (x_1, ..., x_i, ..., x_j, ..., x_n)$, we randomly choose two variables i and j. First, according to the mutation operator for F_S (Section 3) we have to choose a value $p \in (0, 1)$ and a q value depending on p such as $q = \sqrt{(\frac{x_i}{x_j})^2(1 - p^2) + 1}$. By this approach we obtain a feasible solution on the boundary by changing $x_i = x_i \cdot p$ and $x_j = x_j \cdot q$. However, as we did for function F_K, we have to consider the actual value of $\Delta_R(t)$ in the process of finding the appropriate interval for p and hence the value for q. First, two values p_1 and p_2 are obtained such as:

$$x_i \cdot p_1 = V_i \text{ and } V_i = max\{0, x_i - \Delta_R(t)\} \tag{5}$$

and

$$x_j \cdot q = x_j \cdot \sqrt{(\frac{x_i}{x_j})^2(1 - p_2^2) + 1} = V_j \text{ where } V_j = \min\{x_j + \Delta_R(t), 1\} \tag{6}$$

then we set a random value $p \in (\max\{p_1, p_2\}, 1)$ from which we obtain the respective q value. Then, $x_i \cdot p \geq V_i$ and $x_j \cdot q \leq V_j$.

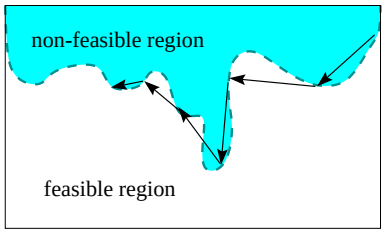

Figure 4: The path is on the boundary of the feasible region

Figure 4 shows a possible path towards to a particular point on boundary of the feasible search space. That path could represent a sequence of feasible points obtained in different cycles of the ACO-B algorithm according to the special boundary operators defined for each test case.

For both functions, we consider two approaches concerning the value of $\Delta_R(t)$[1] for each t. In the first one, called *dynamic approach*, $\Delta_R(t)$ varies according to Eq.1. In the second approach, called *fixed*, the value of $\Delta_R(t)$ is kept constant through the whole running of the algorithm —i.e., $\Delta_R(t) = R$ for $t = 0, \ldots, T$.

5 Experiments and Results

The remaining parameters of ACO-nonB and ACO-B algorithms were set to the following values: $\alpha = 1$, $\eta_i(t) = 1$, for all i and t, i.e., no local information available hence is β not considered (See Eq. 2); $\rho = 0.8$; #ants=20, 50, and 100 released always on 10 alternative directions. For each test case the algorithm run 20 times with different random seed values. Finally, the number of cycles was set to $10,000$ and $30,000$.
The columns in each Table showed represent: n, the number of variables involved in the problem; BK, the Best Known (for F_K) and OV, the optimal value (for F_S) ; the best found value (BF), the average out of 20 runs (avg) and the respective standard deviation ($stdv$). All the tables, except Tables 1 and 2, includes columns for the dynamic and fixed radius approach concerning the boundary search.
Table 1 shows the results obtained by AC0-nonB with 20 ants in $10,000$ cycles for test cases F_S and F_K. It can be observed that ACO-nonB performed better on function F_S than F_K. Also it is important to mention that all solutions found for both test cases were feasible. Further experiments showed that no significant improvement on the performance of ACO-nonB could be reached after $30,000$ cycles. However, by incrementing

n	F_K				F_S			
	BK	BF	avg	stdv	OV	BF	avg	stdv
20	0.8035530	**0.7773011**	0.7155215	0.017	1	**0.9499982**	0.9117287	0.03
50	0.8331937	**0.7479732**	0.7001415	0.013	1	**0.9544134**	0.9280344	0.015
100	NA	**0.7307187**	0.6659797	0.017	1	**0.9005130**	0.8466980	0.037

Table 1: ACO-nonB algorithm runs for $10,000$ cycles (function F_K and F_S). NA: Non Available.

the number of ants (50 and 100) with $10,000$ cycles, an improvement on the performance of the algorithm was obtained. For example, the best values for F_K by using 50 and 100 ants are showed in Table2. For function F_S no further improvement was achieved and the respective results are not showed.

#ants	n (the number of variables)		
	20	50	100
50	0.7869859	0.8094464	0.8020497
100	0.7968856	0.8252584	0.8284985

Table 2: Improved performance of ACO-nonB algorithm with incremental number of ants.

On the other hand, Tables 3 and 5 show the obtained results for functions F_K and F_S respectively by running ACO-B algorithm for $10,000$ cycles.

[1] $R = 10$ for F_K and $R = 1$ for F_S.

n	BK	Dynamic Ratio			Fixed Ratio		
		BF	avg	stdv	BF	avg	stdv
20	0.8035530	**0.8036187**	0.8003203	0.005	**0.8036187**	0.8033460	0.001
50	0.8331937	**0.8352427**	0.8335705	0.001	**0.8352615**	0.8332281	0.002
100	NA	**0.8455312**	0.8427354	0.002	**0.8456039**	0.8416628	0.002

Table 3: ACO-B algorithm runs for $10,000$ cycles (function F_K). NA: Non Available.

The best found values for function F_K (Table 3) with $n = 20, 50$ are better than the best obtained values reported in [13]. Considering the fixed against dynamic search ratio, we can observe that when $n = 20$ both of them get the same best results (0.8036187) . However, when the number of variables get increased ($n = 50$ and $n = 100$), the fixed approach performed a little bit better. Additional experiments by running the ACO-B algorithm for $30,000$ cycles yield some improvements regarding the dynamic approach for $n = 50$ and $n = 100$ variables (Table 4).

n	Dynamic Ratio	Fixed Ratio
20	0.8036190	0.8036187
50	0.8352605	0.8352622
100	0.8456782	0.8456113

Table 4: Some improvements obtained after $30,000$ cycles for function F_k

Concerning function F_S, we observe (Table 5) that either using the dynamic or fixed radius, ACO-B algorithm get very good results. However its performance get a little bit decreased as the number of variables of the problem get increased ($n = 50, 100$). Also it is remarkable the robustness of ACO-B algorithm since the standard deviation is close to 0 for this test case.

n	OV	Dynamic Ratio			Fixed Ratio		
		BF	avg	stdv	BF	avg	stdv
20	1.0	0.9999902	0.9999779	84×10^{-7}	0.9999995	0.9999989	5×10^{-7}
50	1.0	0.9997736	0.9996560	786×10^{-7}	0.9999890	0.9999794	65×10^{-7}
100	1.0	0.9984738	0.9975221	4740×10^{-7}	0.9998920	0.9998246	415×10^{-7}

Table 5: ACO-B algorithm runs for $10,000$ cycles (function F_S)

Similar experiments involving additional number of cycles yielded improved results for function F_S. Table 6 shows how close to the optimum are the best values obtained by ACO-B algorithm.

n	Dynamic Ratio	Fixed Ratio
20	0.9999990	0.9999999
50	0.9999782	0.9999985
100	0.9997975	0.9999863

Table 6: Improved results for function F_S ($30,000$ cycles)

It is important to mention that when ACO-B algorithm uses an increased number of ants, no additional improvements were obtained.

Additionally, in Fig.5 we show the convergence of function F_S (best objective value in each cycle is plotted) from the application of ACO-nonB and ACO-B respectively. It can be observed in the curve for ACO-nonB the oscillation of the objective value around the optimal one. Many of those objetive values correspond to infeasible solutions whereas for ACO-B only feasible solutions are evolved and getting the best value much earlier than the best value obtained by the ACO-nonB version.

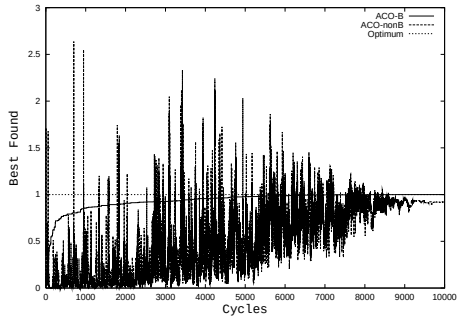

Figure 5: Convergence of ACO-B and ACO-nonB for F_S function.

6 Conclusions

This paper presented an alternative perspective of the ACO approach for continuous constrained spaces based on boundary search. The ACO-B algorithm algorithm outperformed the best values reported for Keane's function ($n = 20$, best known: 0.803553 - $n = 50$, best known: 0.8331937) and performed very well on function F_S. Also, ACO-B showed the importance of boundary operators when is compared against the results obtained by ACO-nonB, i.e., the approach which penalize the infeasible solutions. However, ACO-nonB algorithm performed well for function F_S, but not for function F_K for which was necessary to increment the number of ants.
Our experiments showed the usefulness of the boundary mutation operators taken from the evolutionary techniques in the context of the ACO algorithms. These operators performed very well on the two test cases considered, particularly on function F_K for which new best known values were found.
The performance of ACO-B let us consider the possibility of defining very simple search operators to be applied in the context of the ACO algorithms for constrained numerical optimization problems.

References

[1] Dorigo M., Di Caro G., Gambardella, L.M. (1999) "Ant Algorithms for Discrete Optimization". Artificial Life, 5(2):137-172 (Also available as Tech. Rep. IRIDIA/98-10, Universitè Libre de Bruxelles, Belgium)

[2] Dorigo M. and G. Di Caro (1999). "The Ant Colony Optimization Meta-Heuristic". In D. Corne, M. Dorigo and F. Glover (eds), New Ideas in Optimization. McGraw-Hill, 1999. (Also available as: Tech. Rep. IRIDIA/99-1, Université Libre de Bruxelles, Belgium.)

[3] Bilchev, G. & Parmee, I.C. (1995). "The Ant Colony Metaphor for Searching Continuous Design Spaces". Lectures Notes in Computer Science I993.

[4] Bilchev, G. & Parmee, I.C. (1995). “Natural Self-organizing Systems”. Plymounth Engineering Design Centre Internal Report, PEDC-03-95.

[5] Bilchev, G. & Parmee, I.C. (1996). “Constrained Optimisation with an Ant Colony Search Model”

[6] D. Cornea, M. Dorigo, and F. Glover (1999). “New Ideas in Optimization”. McGraw-Hill International.

[7] Keane, A. (1994). “Genetic Algorithms Digest”, May 19 1994. v8n16.

[8] Leguizamón, G. & Michalewicz, Z. (1999). “A New Version of the Ant System for Subset Problems”. Proceeding of the Congress on Evolutionary Computation. Washington DC, USA. pp 1459-1464.

[9] Michalewicz, Z. (1996). “Genetic Algorithms + Data Structures = Evolution Programs”. 3rd edition, Springer, Berlin.

[10] Michalewicz, Z. & Schoenauer, M. (1996). “Evolutionary Algorithms for Constrained Parameter Optimization”. Evolutionary Computation 4(1):1-32.

[11] Michalewicz, Z., Nazhiyath, G. & Michalewicz, M. (1996). “A Note on Usefulness of Geometrical Crossover for Numerical Optimization Problems”. Proceedings of the 5th Annual Conference on Evolutionary Programming, San Diego, CA, 29 February - 3 March. MIT Press, Cambridge, MA, 1996, pp.305-312.

[12] Reeves, C. (1993). “Modern Heuristic Techniques for Combinatorial Optimization Problems”. Oxford, Blackwell Scientific Publications.

[13] Schoenauer, M. & Michalewicz, Z. (1996). “Evolutionary Computation at the Edge of Feasibility”. Proceedings of the 4th Parallel Problem Solving from Nature, H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel (Editors), Berlin, September 22-27, 1996 Springer-Verlag, Lecture Notes in Computer Science, Vol.1141, pp.245-254.

[14] Schoenauer, M. & Michalewicz, Z. (1998). “Sphere Operators and Their Applicability for Constrained Parameter Optimization Problems”. Evolutionary Programming 1998: 241-250.