An alternative $\mathrm{ACO}_{\mathbb{R}}$ algorithm for continuous optimization problems

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

[1] UMI LAFMIA 3175 CNRS
CINVESTAV-IPN
Departmento de Computación
México D.F., MÉXICO
legui@unsl.edu.ar

[2] CINVESTAV-IPN (Evolutionary Computation Group)
Departamento de Computación
México D.F., MÉXICO
ccoello@cs.cinvestav.mx

Abstract. The Ant Colony Optimization (ACO) metaheuristic embodies a large set of algorithms which have been successfully applied to a wide range of optimization problems. Although ACO practitioners have a long tradition in solving combinatorial optimization problems, many other researchers have recently developed a variety of ACO algorithms for dealing with continuous optimization problems. One of these algorithms is the so-called $\mathrm{ACO}_{\mathbb{R}}$, which is one of the most relevant ACO algorithms currently available for continuous optimization problems. Although $\mathrm{ACO}_{\mathbb{R}}$ has been found to be successful, to the authors' best knowledge its use in high-dimensionality problems (i.e., with many decision variables) has not been documented yet. Such problems are important, because they tend to appear in real-world applications and because in them, diversity loss becomes a critical issue. In this paper, we propose an alternative $\mathrm{ACO}_{\mathbb{R}}$ algorithm ($\mathrm{DACO}_{\mathbb{R}}$) which could be more appropriate for large scale unconstrained continuous optimization problems. We report the results of an experimental study by considering a recently proposed test suite. In addition, the parameters setting of the algorithms involved in the experimental study are tuned using an *ad hoc* tool. Our results indicate that our proposed $\mathrm{DACO}_{\mathbb{R}}$ is able to improve both, the quality of the results and the computational time required to achieve them.

1 Introduction

Several extensions of the Ant Colony Optimization (ACO) metaheuristic [1, 2] for solving continuous problems currently exist. The first ACO extension designed

⋆ On leave of absence from LIDIC - Universidad Nacional de San Luis, San Luis, Argentina.

⋆⋆ The second author is also affiliated to the UMI LAFMIA 3175 CNRS at CINVESTAV-IPN.

to operate on continuous search spaces was introduced by Bilchev et al. [3]. After that, several others were introduced (see [4–11]). The version adopted for our study is the one originally proposed by Socha [12]) and further extended by Socha & Dorigo [13].

In this work, we propose an alternative $\text{ACO}_{\mathbb{R}}$ algorithm for solving large dimensional continuous optimization problems. This study has a recent antecedent (see [14]) in which it was detected that $\text{ACO}_{\mathbb{R}}$ [13] had some limitations when dealing with large dimensional problems. Its main problem was a quick loss of diversity, which had a clear negative impact on the quality of the results achieved by the algorithm. In order to deal with such problem, a simple diversity maintenance mechanism was introduced. Here, we propose a mechanism different to the one adopted in [14], which aims to avoid the loss of diversity by using an alternative mechanism to select the kernels that produce new samples on the search space.

Our experimental study includes a recently proposed test suite of continuous optimization problems which are useful to assess the capacity of an algorithm to deal with large dimensional problems. For determining the parameters setting to be used in the experimental study, we used an automatic tool that approximates a prediction model based on a set of observations (outputs of the real algorithm) for specific design points. Such a proposal intends to be the first step towards improving the ACO metaheuristic in order to achieve a design of a more advanced ACO algorithm which is competitive with respect to other state-of-the-art metaheuristic algorithms used for continuous optimization problems.

The remainder of this paper is organized in the following way: Section 2 briefly describes the original version of the $\text{ACO}_{\mathbb{R}}$ algorithm and Section 3 presents the alternative $\text{ACO}_{\mathbb{R}}$ algorithm (called $\text{DACO}_{\mathbb{R}}$). The section about the experimental study (Section 4) involves three important subsections: Section 4.1 describes the set of test problems adopted; Section 4.2 presents the results of a preliminary study of $\text{ACO}_{\mathbb{R}}$ and $\text{DACO}_{\mathbb{R}}$ on the test suite by using an *ad hoc* tool to tune some selected parameters of the algorithms, and Section 4.3 shows a comparative analysis of the obtained results from algorithms $\text{ACO}_{\mathbb{R}}$ and $\text{DACO}_{\mathbb{R}}$. Finally, in Section 5 we discuss the main achievements of the experimental study. In addition, some lines of future research are also considered.

2 The $\text{ACO}_{\mathbb{R}}$ algorithm

The $\text{ACO}_{\mathbb{R}}$ algorithm was designed with the aim of obtaining a set of *probability density functions* (PDFs). Each PDF is obtained from the search experience and is used to incrementally build a solution $\mathbf{x} \in \mathbb{R}^n$ considering in turn each component x_j ($\forall j = 1 \ldots n$). To approximate a multimodal PDF, Socha & Dorigo [13] proposed a Gaussian kernel which is defined as a weighted sum of several one-dimensional Gaussian functions $g_{ij}(x)$ as follows:

$$G^j(x) = \sum_{i=1}^{k} \omega_i g_{ij}(x) = \sum_{i=1}^{k} \omega_i \frac{1}{\sigma_{ij}\sqrt{2\pi}} e^{-\frac{(x-\mu_{ij})^2}{2(\sigma_{ij})^2}} \tag{1}$$

where $j \in \{1, \ldots, n\}$ identifies the number of dimension, i.e., $\mathrm{ACO}_{\mathbb{R}}$ uses as many Gaussian kernel PDFs as the number of dimensions of the problem. In addition, G^j is parameterized with three vectors: $\boldsymbol{\omega}$, the vector of weights associated with the individual Gaussian functions; $\boldsymbol{\mu}_j$, the vector of means; and $\boldsymbol{\sigma}_j$, the vector of standard deviations. All these vectors have cardinality k, which constitutes the number of Gaussian functions involved.

In $\mathrm{ACO}_{\mathbb{R}}$, a solution archive called T is used to keep track of a number of solutions. The cardinality of archive T is k, that is, the number of kernels that conform the Gaussian kernel. For each solution $\mathbf{x}_i \in \mathbb{R}^n$, $\mathrm{ACO}_{\mathbb{R}}$ maintains the corresponding values of each problem dimension, i.e., $x_{i1}, \ldots, x_{in}$, and the value of the objective function $f(\mathbf{x}_i)$ which are stored satisfying that $f(\mathbf{x}_1) \leq \ldots \leq f(\mathbf{x}_i) \leq \ldots f(\mathbf{x}_k)$. On the other hand, the vector of weights $\boldsymbol{\omega}$ should satisfy that $\omega_1 \geq \ldots \geq \omega_l \geq \ldots \geq \omega_k$. The solutions in T are, therefore, used to dynamically generate probability density functions involved in the Gaussian kernels. More specifically, in order to obtain the Gaussian kernel G^j, the three parameters $\boldsymbol{\omega}$, $\boldsymbol{\mu}_j$, and $\boldsymbol{\sigma}_j$ need to be calculated. Thus, for each G^j, the values of the j-th variable of the k solutions in T become part of the elements of vector $\boldsymbol{\mu}_j$, that is, $\boldsymbol{\mu}_j = (\mu_{1j}, \ldots, \mu_{kj}) = (x_{1j}, \ldots, x_{kj})$. On the other hand, each component of the deviation vector $\boldsymbol{\sigma}_j = (\sigma_{1j}, \ldots, \sigma_{kj})$ is obtained as:

$$\sigma_{ij} = \xi \sum_{e=1}^{k} \frac{|x_{ej} - x_{ij}|}{k-1} \tag{2}$$

where $i \in \{1, \ldots, k\}$ 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 ξ, the lower the convergence speed of the algorithm.

The pheromone update is achieved by considering a set A[3] of size N_a which maintains the newly generated solutions regarding equation (1). The new T (in the next algorithm iteration) is obtained as $T(t+1) = FIRST_k(Rank(T(t) \oplus A(t)))$, i.e., the old solutions in the archive T plus the set of newly created solutions A are ranked and then, the first k best k solutions are selected. In other words, the old solutions compete against the newly generated ones to conform the updated T which maintains its cardinality (k) through the whole search process.

3 The proposed alternative $\mathrm{ACO}_{\mathbb{R}}$ algorithm ($\mathrm{DACO}_{\mathbb{R}}$)

The proposed alternative $\mathrm{ACO}_{\mathbb{R}}$ algorithm is straightforward. The main objective is to keep the diversity as long as possible in order to explore more regions

[3] Set A represents the set of ants according to Socha & Dorigo [13].

of the search space before converging to a possible local optimum from which is usually impossible to escape unless some mechanism is implemented to improve the diversity in the population. Our proposed algorithm, called DACO$_{\mathbb{R}}$ ('D' stands for Diversity) is designed following the same basic principle of ACO$_{\mathbb{R}}$, i.e., from a set of k kernels, a new set of Na solutions is generated via the multimodal kernels (see equation (1)). However, DACO$_{\mathbb{R}}$ always generates $Na = k$ new solutions by considering an alternative approach to select the kernels. More precisely, DACO$_{\mathbb{R}}$ starts from an initial population of k kernels distributed evenly on the whole problem search space. In our case, we have adopted Latin Hypercube Sampling (LHS) under which the search space is divided into k intervals. Figure 1 shows a possible LHS distribution of $k = 5$ kernels on a hypothetical search space of dimension $n = 2$.

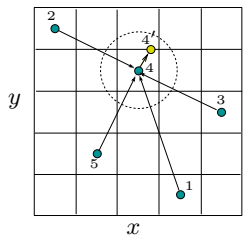


Fig. 1. A possible LHS distribution for $k = 5$ on a hypothetical search space of dimension $n = 2$. The center point in the circle represents the current kernel (number 4) from which a new point is generated by considering the remaining $5 - 1$ kernels.

To generate the new set of solutions $A(t)$ from the actual set of kernels $T(t)$, DACO$_{\mathbb{R}}$ considers two different ways of selecting the kernel from $T(t)$ to produce the corresponding solution in $A(t)$. The first one is as follows: when generating solution i in $A(t)$, the selected kernel from $T(t)$ is the number i, i.e., the solution generated at position i in $A(t)$ will be obtained through a Gaussian distribution with $\boldsymbol{\mu} = \boldsymbol{x}_i$ and a deviation $\boldsymbol{\sigma}$ determined by the remaining set $\{1, \ldots, i-1, i+1, \ldots, k\}$ of $k-1$ kernels in $T(t)$. The newer solution is included in $A(t)$ only if its corresponding objective value is improved with respect to kernel i in $T(t)$; otherwise, the old kernel is copied to $A(t)$ as the new solution generated. In this way, the algorithm behaves as a local explorer around each kernel.

The second approach to generate a solution is by considering the current best kernel in $T(t)$ to generate a particular solution in $A(t)$. Thus, the algorithm globally exploits the best solution in the current population $T(t)$, i.e., the solution generated at position i in $A(t)$ will be obtained through a Gaussian distribution with $\boldsymbol{\mu} = \boldsymbol{x}_{i_{best}}$ and a deviation $\boldsymbol{\sigma}$ determined by the remaining set $\{1, \ldots, i_{best} - 1, i_{best} + 1, \ldots, k\}$ of $k-1$ kernels in $T(t)$. The newer solution is included in $A(t)$ only if its corresponding objective value is improved with respect to kernel i in $T(t)$; otherwise, the old kernel is copied to $A(t)$ as the new solution generated.

Algorithm 1 Outline of $\mathrm{DACO}_{\mathbb{R}}$ algorithm

```
Init_LHS(T);
Get_s( σ );
for t ∈ 1 : t_max do
    A = BuildSolsNew(T, σ);
    T = Sel_Best_one_to_one(T ,A);
    Get_s(σ);
end for
```

To choice between the two ways of constructing $A(t)$ is determined by a parameter q as expressed in equation (3). It should be noticed that parameter q in $\mathrm{DACO}_{\mathbb{R}}$ is different from parameter q in $\mathrm{ACO}_{\mathbb{R}}$. In our $\mathrm{DACO}_{\mathbb{R}}$ algorithm, q determines the way of obtaining some element in $A(t)$ whereas in $\mathrm{ACO}_{\mathbb{R}}$, this parameter determines the relative weight of the ranked kernels. In addition, it also important to note that our $\mathrm{DACO}_{\mathbb{R}}$ algorithm does not need to sort, at each iteration, the set of kernels as required in $\mathrm{ACO}_{\mathbb{R}}$. Parameter ξ is used in $\mathrm{DACO}_{\mathbb{R}}$ in the same way as in $\mathrm{ACO}_{\mathbb{R}}$.

$$A_{ij} = \begin{cases} \text{gen_xj}(T_{ij},\, \sigma_j^i), & q > rand(0,1) \text{ (exploration);} \\ \text{gen_xj}(T_{i_b j},\, \sigma_j^{i_b}), & \text{otherwise (exploitation).} \end{cases}$$
$$\text{where } i = 1, \ldots, N_a \text{ (recall that } N_a = k) \text{ and } j = 1, \ldots, n, \quad \text{and} \quad i_b \text{ is the index to the best current solution in } T(t). \tag{3}$$

Algorithm 1 outlines the main components of $\mathrm{DACO}_{\mathbb{R}}$. Init_LHS() gives the initial set of k kernels through LHS; BuildSolsNew() is in charge of generating $A(t)$ by following the procedure explained before, i.e., either by using a local or a global mechanism. Sel_Best_one_to_one() selects in a one-to-one correspondence the best solutions between $A(t)$ and $T(t)$; and Get_s() obtains the new deviation vectors according to the new populations of kernels recently generated $T(t+1)$.

4 Experimental Study

In this section, we present the experimental study that includes: a) a short description of the test suite adopted to assess the performance of the $\mathrm{ACO}_{\mathbb{R}}$ and $\mathrm{DACO}_{\mathbb{R}}$ algorithms; b) a preliminary study to determine an appropriate parameters setting of the algorithms (for that sake, we have used an automatic tool proposed by Bartz-Beielstein [15] called SPOT [16] (Sequential Parameter Optimization Tool); and c) a comparison between $\mathrm{ACO}_{\mathbb{R}}$ and $\mathrm{DACO}_{\mathbb{R}}$ for the adopted test suite by considering $n \in \{30, 50, 100, 200, 500\}$ dimensions for each of the six problems. All the experiments, except for those corresponding to the preliminary study, were run on a PC having an Intel Pentium (R) 4 processor, running at 3.00Gz, and with 1Gb of RAM. The $\mathrm{ACO}_{\mathbb{R}}$ and $\mathrm{DACO}_{\mathbb{R}}$ algorithms were implemented in the C programming language.

Table 1. Test suite proposed by Tang et al. [17].

Benchmark Problems	Search Range	$f(\mathbf{x}^*)$
$f_1(\mathbf{x}) = \sum_{j=1}^{n} z_j + f_bias_1, \mathbf{z} = \mathbf{x} - \mathbf{o}$ $\mathbf{o} = (o_1, o_2, \ldots, o_n)$; the shifted global optimum	[-100,100]	-450
$f_2(\mathbf{x}) = max_j\{\lvert z_j \rvert, 1 \leq j \leq n\} + f_bias_2, \mathbf{z} = \mathbf{x} - \mathbf{o}$ $\mathbf{o} = (o_1, o_2, \ldots, o_n)$; the shifted global optimum	[-100,100]	-450
$f_3(\mathbf{x}) = \sum_{j=1}^{n-1}(100 \cdot (z_j^2 - z_j)^2 + (z_j - 1)^2) + f_bias_3,$ $\mathbf{z} = \mathbf{x} - \mathbf{o} + \mathbf{1}$; $\mathbf{o} = (o_1, o_2, \ldots, o_n)$; the shifted global optimum	[-100,100]	390
$f_4(\mathbf{x}) = \sum_{j=1}^{n}(z_j^2 - 10 \cdot \cos(2\pi z_j) + 10) + f_bias_4, \mathbf{z} = \mathbf{x} - \mathbf{o}$ $\mathbf{o} = (o_1, o_2, \ldots, o_n)$; the shifted global optimum	[-5,5]	-330
$f_5(\mathbf{x}) = \sum_{j=1}^{n} \frac{z_j^2}{4000} - \prod_{j=1}^{n} \cos(\frac{z_j}{\sqrt{j}}) + 1 + f_bias_5, \mathbf{z} = \mathbf{x} - \mathbf{o}$ $\mathbf{o} = (o_1, o_2, \ldots, o_n)$; the shifted global optimum	[-600,600]	-180
$f_6(\mathbf{x}) = -20\exp(-0.2\sqrt{\frac{1}{n}\sum_{j=1}^{n} z_j^2})$ $- \exp(\frac{1}{n}\sum_{j=1}^{n}\cos(2\pi z_j)) + 20 + f_bias_6),$ $\mathbf{z} = \mathbf{x} - \mathbf{o}$; $\mathbf{o} = (o_1, o_2, \ldots, o_n)$; the shifted global optimum	[-32,32]	-140

4.1 The Adopted Test Suite

We selected 6 problems from the benchmark functions prepared for the "Special Session and Competition on Large Scale Global Optimization" at the *2008 IEEE Congress on Evolutionary Computation (CEC'08)* [17]. The problems represent a set of scalable functions for high-dimensional optimization. See Table 4.1 for a description of these problems and their corresponding optimum values. Particularly, the objective of this special session was to bring to the research community newer and more challenging problems to assess current nature-inspired optimization algorithms as well as other, novel optimization algorithms.

4.2 Parameters Settings for $\text{ACO}_{\mathbb{R}}$ and $\text{DACO}_{\mathbb{R}}$

In order to conduct the preliminary study and establish the most appropriate parameters setting for our proposed approach, it was necessary the integration of the algorithms $\text{ACO}_{\mathbb{R}}$ and $\text{DACO}_{\mathbb{R}}$ (implemented in C) with SPOT (implemented in MATLAB) through the compiler MEX. After tuning the corresponding parameters, all the algorithms ran as standalone processes in the usual way.

As an optimization algorithm, SPOT includes several specific parameters that must be provided when applied to a particular algorithm. In our case, we used the default parameters setting for this tool (e.g., the sampling procedure applied is the Latin Hypercube Sampling where the number of design points is set to 16 by default). Additionally, SPOT needs to run the algorithm (either $\text{ACO}_{\mathbb{R}}$ or $\text{DACO}_{\mathbb{R}}$) to fit a model based on a sample of observations; accordingly, some fixed parameters (not included in the algorithm's design, explained below) of the respective algorithm under study need to be provided. For example: the problem dimension ($n = 100$ was the chosen setting), the maximum number of iterations (was set to $t_{max} = 1000$), and the number of kernels and ants (they were set

respectively to $k = 50$ and $N_a = 50$). It should be noticed that for $\mathrm{DACO}_{\mathbb{R}}$, Na is always set as k (i.e., $Na = k$). In addition, the setting for the number of dimensions and maximum number of iterations, was only used to calibrate the selected parameters (q and ξ) as explained next. For the comparative study (see Section 4.3) the respective settings for these parameters are different (except for k and N_a).

The definition of the *Problem Design* (X_P regarding the terminology taken from [15]) for both algorithms includes function F which is expressed as:

$$F(\mathbf{x}, n) = \sum_{i=1}^{6} (f_i(\mathbf{x}, n) - f_i^*)/f_i^* \tag{4}$$

where f_i represents one of the six functions from the test suite studied (presented in Table 4.1), n is the problem's dimensionality (all these functions are scalable), and f_i^* represents the optimal value for function i (it must be noticed that for any dimension n, the optimal values remain the same). Thus, F expresses the summation of the percentage error over the six functions. In this way, we apply SPOT as considering only one problem in the process of tuning the corresponding algorithms' parameters.

We first define the *Algorithm Design* for $\mathrm{DACO}_{\mathbb{R}}$ (see [15]) as $X_{\mathrm{DACO}_{\mathbb{R}}}$, the region determined by parameters q and ξ as follows: $0 \leq q \leq 1$ and $0 \leq \xi \leq 1$. After the initial application of SPOT to calibrate these parameters, we observed that the regions determined by $q \in (0.5, 1]$ and $\xi \in [0, 0.5)$ can be eliminated from the experimental study due to the poor performance of the algorithm for such parameter values. Accordingly, we redefined $X_{\mathrm{DACO}_{\mathbb{R}}}$ as the region determined by $0 \leq q \leq 0.5$ and $0.5 \leq \xi \leq 1$. The best parameters setting for $\mathrm{DACO}_{\mathbb{R}}$ was: $q = 0.1172$ and $\xi = 0.6063$. Figure 2 shows the output from SPOT for the regions considered of the algorithm's design ($X_{\mathrm{DACO}_{\mathbb{R}}}$) with respect to parameters q and ξ. On the left, we can observe the response surface of the predicted values (PV) of function F for $\mathrm{DACO}_{\mathbb{R}}$. On the right, we show the corresponding surface of the Mean Square Error (MSE).

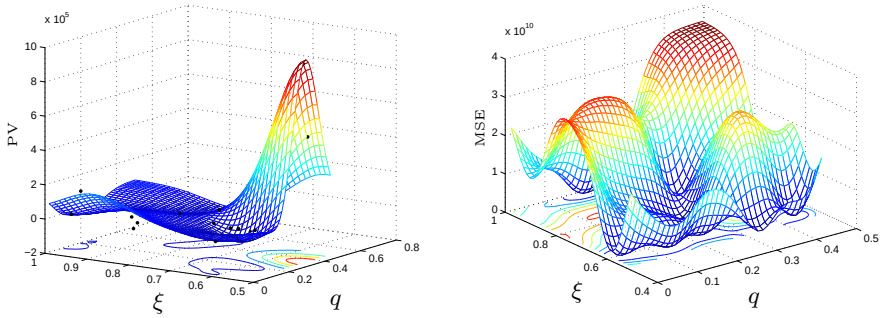
Similarly, for $\mathrm{ACO}_{\mathbb{R}}$, $X_{\mathrm{ACO}_{\mathbb{R}}}$ was determined by considering the region $0 \leq q \leq 0.5$ and $0.5 \leq \xi \leq 1$ from which SPOT reported $q = 0.0103$ and $\xi = 0.8257$ as the best corresponding parameters for the $\mathrm{ACO}_{\mathbb{R}}$ algorithm.

Finally, it must be remarked that for both algorithms ($\mathrm{ACO}_{\mathbb{R}}$ and $\mathrm{DACO}_{\mathbb{R}}$), the initial population of kernels was obtained from input files previously generated by using the MATLAB function `lhsdesign`.

4.3 Performance of $\mathrm{ACO}_{\mathbb{R}}$ and $\mathrm{DACO}_{\mathbb{R}}$ on the selected problems

This section presents a comparative study of $\mathrm{ACO}_{\mathbb{R}}$ and $\mathrm{DACO}_{\mathbb{R}}$ on the six problems presented in Section 4.1. The parameters settings for these algorithms are as follows. For both algorithms, $k = Na = 50$, t_{max} was set respectively to 6000, 10000, 14000, 20000, 40000, and 100000 when running the algorithms with problems of dimensions $n = 30$, $n = 50$, $n = 70$, $n = 100$, $n = 200$, and $n = 500$. These values for t_{max} were selected considering the criteria followed in [17].

Fig. 2. Regions of the algorithm's design ($X_{\mathrm{DACO}_{\mathbb{R}}}$) with respect to parameters q and ξ. (Left)- response surface of the predicted values (PV) of function F for $\mathrm{DACO}_{\mathbb{R}}$ and (Right)- the corresponding surface of the Mean Square Error (MSE).



Particularly, we considered twice the maximum number of function evaluations (FES) proposed for dimension $n = 100$. Accordingly, we obtained the corresponding t_{max} values proportionally for each dimension studied. Under these parameters settings, both algorithms ran for the same number of function evaluations for each of the problems and dimensions considered. With respect to the remaining parameters (i.e., q and ξ), the used values were those reported by SPOT which are, for $\mathrm{DACO}_{\mathbb{R}}$, $q = 0.1172$ and $\xi = 0.6063$; whereas for $\mathrm{ACO}_{\mathbb{R}}$, are $q = 0.0103$ and $\xi = 0.8257$. Both algorithms were run considering 25 random seeds for each combination of problem, dimension, and parameters setting.

The results are shown in all the figures displayed in the Appendix. We adopted boxplots to show the distribution of the results expressed as the percentage of the error with respect to the optimum values. We divided the presentation of the results based on the problem's dimension. On the one hand, Figures 3 and 4 show respectively the results for dimensions $n = 30, 50$ and $n = 70, 100$. On the other hand, and because of the large differences found in the percentage error for the larger dimensions considered in this work ($n = 200$ and $n = 500$), we split the presentation in three figures for: i) problems 1,2,4, and 5 with $n = 200, 500$ (Figure 5); ii) problems 3 and 6 with $n = 200$ (Figure 6); and iii) problems 3 and 6 with $n = 500$ (Figure 7). The x-axis in each figure indicates the problem number whereas the corresponding super-index represents the corresponding applied algorithm (a stands for $\mathrm{ACO}_{\mathbb{R}}$ and d stands for $\mathrm{DACO}_{\mathbb{R}}$). The non-parametric Mann-Whitney-Wilcoxon test at a level of 5% of confidence was applied to assess the significance on the differences on the corresponding medians of $\mathrm{DACO}_{\mathbb{R}}$ with respect to $\mathrm{ACO}_{\mathbb{R}}$. Thus, a p-value < 0.05 indicates that based on the median values $\mathrm{DACO}_{\mathbb{R}}$ outperforms $\mathrm{ACO}_{\mathbb{R}}$. It is worth mentioning that the statistical test was applied considering, for each dimension, a sample of 25×6 points (i.e., all the percentage error values for each problem and run were collected as one sample). The p-values are respectively $4.0459e-007$, $3.7094e-0051$, 0.0083, 0.0117, and $2.6185e-005$ for dimensions 30, 50, 70, 100, 200, and 500. From

this statistical point of view, $DACO_{\mathbb{R}}$ outperforms $ACO_{\mathbb{R}}$ for all problems and dimensions considered. More precisely, when taking into account the shape and location of the boxplots for all dimensions and considering one problem in turn, we can observe a similar behavior of both algorithms. On the one hand, the algorithms scale fairly well with larger dimensions for problems 1, 2, and 5. Also for these problems, it can be seen that both algorithms preformed robustly and achieved high quality results (mainly for dimensions $n \in \{30, 50, 70, 100\}$). However, $DACO_{\mathbb{R}}$ found the best results for these problems with all the dimensions tested. When increasing the problems' dimensionality for problems 1,2, and 5 (i.e., $n = 200$ and $n = 500$) we found a less robust behavior and results of lower quality for algorithm $ACO_{\mathbb{R}}$. On the other hand, problems 3, 4, and 6 represent a challenge for both algorithms. It can be clearly observed that there was a large increase in the percentage error as the problem dimensionality increased. Although both algorithms have difficulties to solve and scale on these three problems, the behavior of $DACO_{\mathbb{R}}$ is superior to that of $ACO_{\mathbb{R}}$. Finally, is it worth remarking that $DACO_{\mathbb{R}}$ needed less CPU time to complete the same number of function evaluations than the $ACO_{\mathbb{R}}$ algorithm. Indeed, our proposed approach required about 25% less CPU time, on average, than $ACO_{\mathbb{R}}$, for all the problems and dimensions tested in our study. This can be explained based on the fact that our proposed $DACO_{\mathbb{R}}$ algorithm does not include the sorting procedure used by the original $ACO_{\mathbb{R}}$ to produce, at each iteration, a ranked set of kernels. In $DACO_{\mathbb{R}}$, it is only necessary to maintain an index to the current best kernel (i_{best} in equation (3)).

5 Discussion and Conclusions

In this work we presented $DACO_{\mathbb{R}}$, an alternative to the $ACO_{\mathbb{R}}$ algorithm for dealing with large dimensional continuous problems. The achieved results show the potential of our proposed $DACO_{\mathbb{R}}$ algorithm to solve large scale optimization problems. Our results lead us to think about other possible modifications, including more sophisticated mechanisms to control the intensification and the diversification during the search. This could strengthen the position of the ACO metaheuristic with respect to state-of-the-art algorithms in current use for large dimensional continuous optimization problems (e.g., differential evolution and evolution strategies). Our future work will include the incorporation of a mechanism to better control the region of local exploration as well as to automatically decide between a global and a local exploration based on the diversity observed in the current population of kernels. Improved versions of $DACO_{\mathbb{R}}$ should certainly be compared with some state-of-the-art metaheuristics for continuous optimization problems. Additionally, it is important to acquire more experience in the use of tools (either SPOT or some other approach) to appropriately set the main parameters of the future version of the algorithms to be studied.

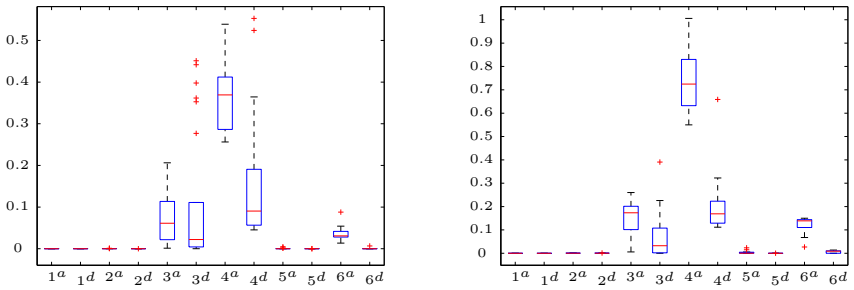
Acknowledgments The first author acknowledges the support from the UMI-LAFMIA 3175 CNRS at CINVESTAV-IPN and from the Universidad Nacional

de San Luis, Argentina. The second author gratefully acknowledges support from CONACyT project no 103570.

References

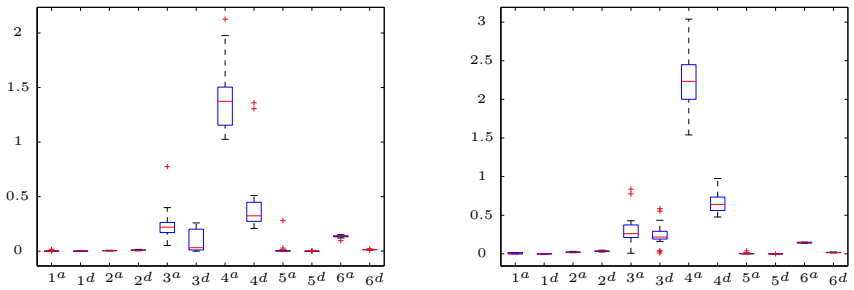
1. Corne, D., Dorigo, M., Glover, F., eds.: New Ideas in Optimization. McGraw-Hill International, London, UK (1999)
2. Dorigo, M., Stützle, T.: Ant Colony Optimization. Mit-Press (2004)
3. Bilchev, G., Parmee, I.: The Ant Colony Metaphor for Searching Continuous Design Spaces. In Fogarty, T.C., ed.: Evolutionary Computing. AISB Workshop. Springer, Sheffield, UK (April 1995) 25–39
4. Monmarché, N., Venturini, G., Slimane, M.: On how pachycondyla apicalis ants suggest a new search algoritm. Future Generation Computer Systems **16** (2000) 937–946
5. 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), Brussels, Belgium, Springer Verlag. Lecture Notes in Computer Science Vol. 2463 (2002) 288–289
6. Dréo, J., Siarry, P.: A New Ant Colony Algorithm Using the Heterarchical Concept Aimed at Optimization of Multiminima Continuous Functions. In Dorigo, M., Di Caro, G., Sampels, M., eds.: Proceedings of the Third international Workshop on Ant Algorithms - ANTS 2002. Springer-Verlag. Lecture Notes in Computer Science Vol. 2463, Brussels, Belgium (September 2002) 216–221
7. Dréo, J., Siarry, P.: Continuous Interacting Ant Colony Algorithm Based on Dense Heterarchy. Future Generation Comp. Syst. **20**(5) (2004) 841–856
8. Ling Chen, J. Shen, L.Q., Chen, H.: An improved ant colony algorithm in continuous optimization. Journal of Systems Science and Systems Engineering **12**(2) (2003) 224–235
9. Pourtakdoust, S., 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.: Proceedings of Ant Colony Optimization and Swarm Intelligence, 4th International Workshop, ANTS Workshop 2004, Brussels, Belgium, Springer-Verlag (2004) 294–301. Lecture Notes in Computer Science Vol. 3172
10. Kong, M., Tian, P.: A direct application of ant colony optimization to function optimization problem in continuous domain. In: ANTS Workshop. (2006) 324–331
11. Hu, X., Zhang, J., Li, Y.: Orthogonal methods based ant colony search for solving continuous optimization problems. J. Comput. Sci. Technol. **23**(1) (2008) 2–18
12. Socha, K.: ACO for continuos and mixed-variable optimization. In Dorigo, M., Birattari, M., Blum, C., Gambardella, L.M., Mondada, F., Stützle, T., eds.: Proceedings of Ant Colony Optimization and Swarm Intelligence, 4th International Workshop, ANTS Workshop 2004, Brussels, Belgium, Springer-Verlag. Lecture Notes in Computer Science Vol. 3172 (2004) 25–36
13. Socha, K., Dorigo, M.: Ant colony optimization for continuous domains. European Journal of Operational Research **185**(3) (2008) 1155–1173
14. Leguizamón, G., Coello Coello, C.A.: A Study of the Scalability of $ACO_{\mathbb{R}}$ for Continuous Optimization Problems. Technical Report EVOCINV-01-2010, Evolutionary Computation Group at CINVESTAV, Departamento de Computación, CINVESTAV-IPN, México (February 2010)

15. Bartz-Beielstein, T.: Experimental Research in Evolutionary Computation: The New Experimentalism (Natural Computing Series). Springer-Verlag New York, Inc., Secaucus, NJ, USA (2006)
16. Bartz-Beielstein, T., Preuss, M.: Spot (sequential parameter optimization tool). http://www.gm.fh-koeln.de/campus/personen/lehrende/thomas.bartz-beielstein/00489/
17. Tang, K., Yao, X., Suganthan, P.N., MacNish, C., Chen, Y.P., Chen, C.M., Yang, Z.: Benchmark Functions for the CEC'2008 Special Session and Competition on Large Scale Global Optimization. Technical report, Nature Inspired Computation and Applications Laboratory, USTC, China (2007)



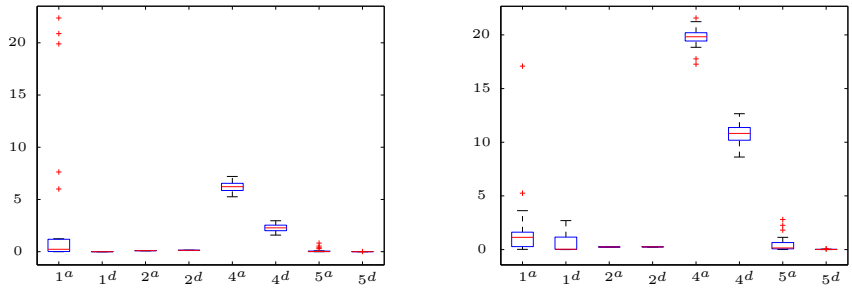
(a) Dimension $n = 30$ (b) Dimension $n = 50$

Fig. 3. Percentage error for the six benchmark problems with dimension $n \in \{30, 50\}$.



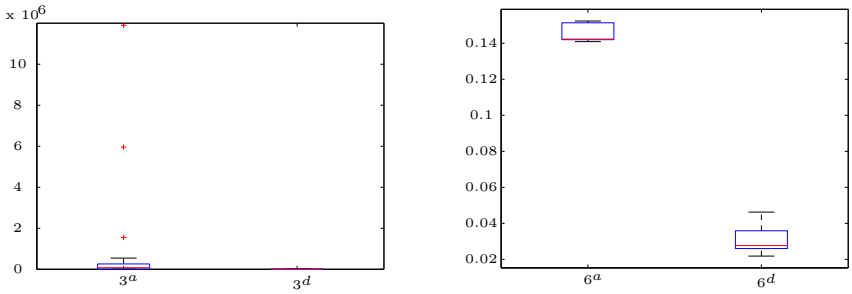
(a) Dimension $n = 70$ (b) Dimension $n = 100$

Fig. 4. Percentage error for the six benchmark problems with dimension $n \in \{70, 100\}$.



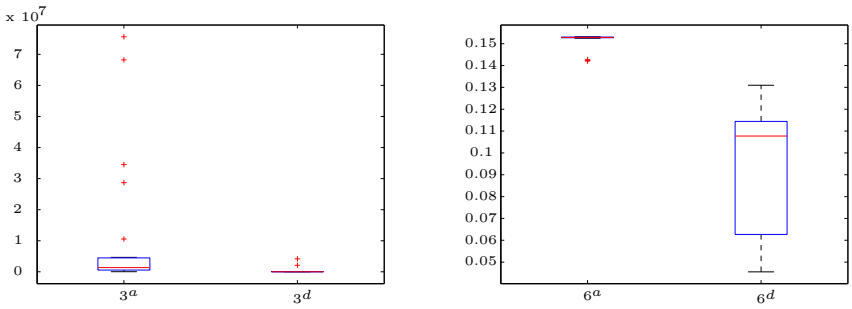
(a) Dimension $n = 200$ (b) Dimension $n = 500$

Fig. 5. Percentage error for problems 1, 2, 4, and 5 with dimension $n \in \{200, 500\}$.



(a) Dimension $n = 200$, problem 3 (b) Dimension $n = 200$, problem 6

Fig. 6. Percentage error for problems 3 and 6 with dimension $n = 200$.



(a) Dimension $n = 500$, problem 3 (b) Dimension $n = 500$, problem 6

Fig. 7. Percentage error for problems 3 and 6 with dimension $n = 500$.